

UTILITY PATENT APPLICATION TRANSMITTAL (Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.
Q99-1113-US1

Total Pages in this Submission

TO THE ASSISTANT COMMISSIONER FOR PATENTSBox Patent Application
Washington, D.C. 20231

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new utility patent application for an invention entitled:

DATA CHECKSUM METHOD AND APPARATUS

and invented by:

RODNEY VAN METER III
 11/12/99
 09/439776
 11/12/99
If a **CONTINUATION APPLICATION**, check appropriate box and supply the requisite information:
☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: _____

Enclosed are:

Application Elements

1. ☒ Filing fee as calculated and transmitted as described below
2. ☒ Specification having 47 pages and including the following:
 - a. ☒ Descriptive Title of the Invention
 - b. ☐ Cross References to Related Applications (if applicable)
 - c. ☐ Statement Regarding Federally-sponsored Research/Development (if applicable)
 - d. ☐ Reference to Microfiche Appendix (if applicable)
 - e. ☒ Background of the Invention
 - f. ☒ Brief Summary of the Invention
 - g. ☒ Brief Description of the Drawings (if drawings filed)
 - h. ☒ Detailed Description
 - i. ☐ Claim(s) as Classified Below
 - j. ☒ Abstract of the Disclosure

UTILITY PATENT APPLICATION TRANSMITTAL (Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.
Q99-1113-US1

Total Pages in this Submission

Application Elements (Continued)

3. ☒ Drawing(s) (when necessary as prescribed by 35 USC 113)
- a. ☐ Formal Number of Sheets _____
- b. ☒ Informal Number of Sheets 15
4. ☒ Oath or Declaration
- a. ☒ Newly executed (original or copy) ☐ Unexecuted
- b. ☐ Copy from a prior application (37 CFR 1.63(d)) (for continuation/divisional application only)
- c. ☒ With Power of Attorney ☐ Without Power of Attorney
- d. ☐ DELETION OF INVENTOR(S)
Signed statement attached deleting inventor(s) named in the prior application,
see 37 C.F.R. 1.63(d)(2) and 1.33(b).
5. ☐ Incorporation By Reference (usable if Box 4b is checked)
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. ☐ Computer Program in Microfiche (Appendix)
7. ☐ Nucleotide and/or Amino Acid Sequence Submission (if applicable, all must be included)
- a. ☐ Paper Copy
- b. ☐ Computer Readable Copy (identical to computer copy)
- c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

Accompanying Application Parts

8. ☒ Assignment Papers (cover sheet & document(s))
9. ☒ 37 CFR 3.73(B) Statement (when there is an assignee)
10. ☐ English Translation Document (if applicable)
11. ☐ Information Disclosure Statement/PTO-1449 ☐ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Acknowledgment postcard
14. ☒ Certificate of Mailing
- ☐ First Class ☒ Express Mail (Specify Label No.): EL111015975US

UTILITY PATENT APPLICATION TRANSMITTAL
(Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.
Q99-1113-US1

Total Pages in this Submission

Accompanying Application Parts (Continued)

15. ☐ Certified Copy of Priority Document(s) (if foreign priority is claimed)

16. ☒ Additional Enclosures (please identify below):

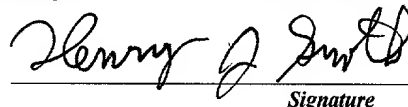
APPENDIX A - 3 PAGES
APPENDIX B - 7 PAGES

Fee Calculation and Transmittal

CLAIMS AS FILED

For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	29	- 20 =	9	x \$18.00	\$162.00
Indep. Claims	3	- 3 =	0	x \$78.00	\$0.00
Multiple Dependent Claims (check if applicable) <input type="checkbox"/>					\$0.00
BASIC FEE					\$760.00
OTHER FEE (specify purpose) ASSIGNMENT RECORDAL					\$40.00
TOTAL FILING FEE					\$962.00

- ☐ A check in the amount of _____ to cover the filing fee is enclosed.
- ☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. **50-0283** as described below. A duplicate copy of this sheet is enclosed.
- ☒ Charge the amount of **\$962.00** as filing fee.
 - ☒ Credit any overpayment.
 - ☒ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
 - ☐ Charge the issue fee set in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).


Signature

HENRY J. GROTH, REG. NO. 39,696
QUANTUM CORPORATION
PATENT DEPARTMENT
500 MCCARTHY BLVD.
MILPITAS, CA 95035
408/894-5425

Dated: **NOVEMBER 12, 1999**

cc:

Data Checksum Method and Apparatus

By

Rodney Van Meter III

Notice of Inclusion of Copyrighted Material

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

Field of the Invention

The present invention relates to providing data checksums, and in particular to providing checksums for data retrieved from data storage devices.

Background of the Invention

Data processing systems typically include several data processors interconnected via a network for data communication, wherein one or more of the data processors include at least one data storage device. Upon a request from a requester processor, data is retrieved from a data storage device of a data processor and a checksum is generated for the retrieved data. The retrieved data and checksum are then transferred to the requester, wherein the requester computes a checksum for the received data and compares it to the received checksum to detect end-to-end data transfer errors.

A major disadvantage of conventional techniques for generating checksums is that such checksums are calculated using software routines as a separate pass over the data retrieved from the data storage device before the data is transferred to the

1 requester. Specifically, the software checksum calculation occurs after the retrieved
2 data is stored in a buffer memory, and before the retrieved data is transferred to the
3 requester. The process for performing the software checksum is executed on a
4 processor unit, such as a 32-bit central processing unit (CPU). The overhead for
5 software checksum calculation can be for example nineteen CPU instructions to
6 checksum thirty two bytes of data. As such, the software checksum calculation
7 consumes precious computing time and resources. Such high overhead degrades data
8 communication response, and negatively impacts the performance of data processing
9 systems.

10
11 There is, therefore, a need for a method and apparatus to efficiently determine
12 checksums for data retrieved from data storage devices.

13 **Brief Summary of the Invention**

14
15 The present invention satisfies these needs. In one embodiment, the present
16 invention provides a method of generating checksum values for data segments
17 retrieved from a data storage device for transfer into a buffer memory. The method
18 includes the steps of maintaining a checksum list comprising a plurality of entries
19 corresponding to the data segments stored in the buffer memory, each entry being for
20 storing a checksum value for a corresponding data segment stored in the buffer
21 memory. The method further includes the steps of, for each data segment retrieved
22 from the storage device: calculating a checksum value for that data segment using a
23 checksum circuit; providing an entry in the checksum list corresponding to that data
24 segment; storing the checksum value in the selected entry in the checksum list; and
25 storing that data segment in the buffer memory. Preferably, the step of calculating the
26 checksum further includes the steps of calculating the checksum for that data segment
27 using the checksum circuit as the data segment is transferred into the buffer memory.
28

1 For transferring packets of data out of the buffer memory, a checksum value can
2 be calculated for data in each packet based on one or more checksum values stored in
3 the checksum list. Providing a checksum value for each packet includes the steps of
4 calculating the checksum value for that packet as a function of at least one checksum
5 value stored in the checksum list. In one case, each packet in a set of the packets
6 includes one or more data segments; and the steps of providing a checksum value for
7 each of the packets in the set of packets further includes the steps of: retrieving the
8 checksum value for each data segment in that packet from a corresponding entry in
9 the check list; and calculating a checksum value for that packet as a function of the
10 retrieved checksum values.

11
12 In another case, each packet in a set of the packets includes: (i) one or more
13 complete data segments, and (ii) a fragment of each of one or more data segments.
14 Providing a checksum value for each of the packets further includes the steps of:
15 retrieving the checksum value for each of the one or more complete data segments in
16 that packet from a corresponding entry in the checksum list; calculating an intermediate
17 checksum value as a function of the retrieved checksum values; for each fragment of a
18 data segment in that packet, calculating a checksum value for data in that data
19 segment fragment; and adjusting the intermediate checksum value using the checksum
20 values for said one or more data segment fragments to provide a checksum value for
21 all of the data in that packet.

22
23 Yet in another case, each packet in a set of the packets includes: (i) at least one
24 complete data segment, and (ii) a fragment of each of one or more data segments.
25 Providing a checksum value for data in each of the packets in the set of packets further
26 includes the steps of: retrieving the checksum value for the at least one complete data
27 segment in that packet from a corresponding entry in the checksum list; for each of the
28 one or more data segment fragments in the packet, retrieving the checksum value from

1 the entry in the checksum list corresponding to the data segment of which that data
2 segment fragment is a part, and calculating a checksum value for data in a
3 complementary fragment of that data segment; calculating an intermediate checksum
4 value as a function of the retrieved checksum values; and adjusting the intermediate
5 checksum value using the checksum values for said one or more complementary data
6 segment fragments to provide a checksum value for all of the data in that packet.

7
8 In another aspect, the present invention provides a storage device comprising
9 storage media for storing data segments; buffer memory; and a checksum circuit for
10 providing checksum values for data segments retrieved from the storage media for
11 transfer into the buffer memory, and for storing the checksum values in a checksum list
12 including a plurality of entries corresponding to the data segments stored in the buffer
13 memory, each entry for storing a checksum value for a corresponding data segment
14 stored in the buffer memory. As such, for transferring packets of data out of the buffer
15 memory, a checksum value can be calculated for data in each packet based on one or
16 more checksum values stored in the checksum list. The checksum circuit comprises a
17 logic circuit for calculating a checksum value for a data segment; means for locating an
18 entry in the checksum table corresponding to that data segment; and means for storing
19 the checksum value in the located entry in the checksum list. The storage device can
20 further comprise a microcontroller configured by program instructions according to the
21 method of the present invention for transferring packets of data out of the buffer
22 memory, and for providing a checksum value for data in each packet.

23
24 The present invention provides significant performance improvements for data
25 service from storage devices such as disk or tape drives through a buffer memory onto
26 a network. Calculating checksums using a checksum circuit as data is transferred into
27 a buffer memory substantially reduces memory bandwidth consumed and the number
28 CPU instructions cycles needed for software checksums. Further, data transfer latency

for checksums is reduced.

Brief Description of the Drawings

These and other features, aspects and advantages of the present invention will become understood with reference to the following description, appended claims and accompanying figures where:

FIG. 1 shows a block diagram of the architecture of an embodiment of a computer system including a disk storage system according to an aspect of the present invention;

FIG. 2 shows a block diagram of the architecture of an embodiment of the drive electronics of FIG. 1;

FIG. 3 shows a block diagram of the architecture of an embodiment of the data controller of FIG. 2 including an example checksum circuit, receiving data from data disks through a read channel;

FIG. 4 shows a flow diagram of an embodiment of steps for calculating checksum values in the data controller of FIG. 3 according to the present invention;

FIGS. 5A-C show examples of packaging of data segments in packets for transfer out of the buffer memory;

FIG. 6 shows a flow diagram of an embodiment of a process for providing checksum values for packets of one or more complete data segments;

FIG. 7A shows a flow diagram of an embodiment of a process for providing checksum values for packets of at least one complete data segment and a fraction one or more data segments;

FIG. 7B shows a flow diagram of another embodiment of a process for providing checksum values for packets of at least one complete data segment and a fraction one or more data segments;

FIG. 7C shows a flow diagram of yet another embodiment of a process for providing checksum values for packets of at least one complete data segment and a

1 fraction one or more data segments;

2 FIG. 8 shows a flow diagram of an embodiment of a process for providing
3 checksum values for packets of one more data segment fragments;

4 FIG. 9 shows a flow diagram of an embodiment of a process for packaging data
5 segments into packets, and calculating checksums for the packets;

6 FIGS. 10A-C illustrate example layouts for a TCP/IP packet in an Ethernet
7 frame, an IP header with normal fields, and a pseudo header generated from the IP
8 header, respectively;

9 FIG. 11 is a table of checksums for data segment fragments in packets;

10 FIG. 12 is a block diagram of the architecture of another embodiment of the data
11 controller of FIG. 2 including an example checksum circuit, receiving data from data
12 disks through a read channel;

13 FIG. 13 is a block diagram of the architecture of an embodiment of the checksum
14 circuit of FIG. 12;

15 FIG. 14 is a block diagram of an embodiment of the I/O adapter of FIG. 1
16 including an example checksum circuit according to another aspect of the present
17 invention; and

18 FIG. 15 is a block diagram of an embodiment of the network interface device of
19 FIG. 1 including an example checksum circuit according to another aspect of the
20 present invention.

21 22 **Detailed Description of the Invention**

23 Referring to FIG. 1, an example computer system 10 is shown to include a
24 central processing unit ("CPU") 14, a main memory 16, and I/O bus adapter 18, all
25 interconnected by a system bus 20. Coupled to the I/O bus adapter 18 is an I/O bus 22
26 that can be e.g. a small computer system interconnect (SCSI) bus, and which supports
27 various peripheral devices 24 including a disk storage unit such as a disk drive 25. The
28 disk drive 25 includes drive electronics 26 and a head disk assembly 28 ("HDA"). As

1 shown in FIG. 12, in one embodiment, the HDA 28 can include data disks 29 rotated by
2 a spindle motor 23, and transducers 27 moved radially across the data disks 29 by one
3 or more actuators 37 for writing data to and reading data from the data disks 29.

4
5 Referring to FIG. 2, in one embodiment, the drive electronics 26 of FIG. 1 is
6 shown to include a data controller 30 interconnected to a servo controller 34 via bus 31,
7 and a read/write channel 32 interconnected to the data controller 30 via a data buffer
8 bus 33. Head positioning information from data disks are induced into the transducers,
9 converted from analog signals to digital data in the read/write channel 32, and
10 transferred to the servo controller 34, wherein the servo controller 34 utilizes the head
11 positioning information for performing seek and tracking operations of the transducers
12 27 over the disk tracks.

13
14 A typical data transfer initiated by the CPU 14 to the disk drive 25 may involve for
15 example a direct memory access ("DMA") transfer of digital data from the memory 16
16 onto the system bus 20 (FIG. 1). Data from the system bus 20 are transferred by the
17 I/O adapter 18 onto the I/O bus 22. The data are read from the I/O bus 22 by the data
18 controller 30, which formats the data into data segments with the appropriate header
19 information and transfers the data to the read/write channel 32. The read/write channel
20 32 operates in a conventional manner to convert data between the digital form used by
21 the data controller 30 and the analog form suitable for writing to data disks by
22 transducers in the HDA 28.

23
24 For a typical request for transfer of data segments from the HDA 28 to the CPU
25 14, the data controller 30 provides a disk track location to the servo controller 34 where
26 the requested data segments are stored. In a seek operation, the servo controller 34
27 provides control signals to the HDA 28 for commanding an actuator 37 to position a
28 transducer 27 over said disk track for reading the requested data segments therefrom.

1 The read/write channel 32 converts the analog data signals from the transducer 37 into
2 digital data and transfers the data to the data controller 30. The data controller 30
3 places the digital data on the I/O bus 22, wherein the I/O adapter 18 reads the data
4 from the I/O bus 22 and transfers the data to the memory 16 via the system bus 20 for
5 access by the CPU 14.

6
7 Referring to FIG. 3, to provide checksum values for the requested data
8 segments, in one embodiment the data controller 30 includes a checksum circuit 36
9 and a buffer memory 38 such as RAM for storing data segments 48 retrieved from data
10 disks 29 in the HDA 28. Data signals from disks 29 are converted into digital data by
11 the read channel 32. The digital data are blocked as data segments 48 of the same
12 size for transfer into the memory 38. Generally, the checksum circuit 36 calculates
13 checksum values for data segments 48 and stores the checksum values into the
14 checksum list or table 42 in memory, such as in a designated part of the typical disk
15 buffer RAM 38.

16
17 Referring to FIG. 4 in conjunction with FIG. 3, in one example checksum process
18 according to the present invention, the checksum list 42 is maintained for storing
19 checksum values (step 50). The checksum list 42 comprises e.g. a plurality of entries
20 44 corresponding to the data segments 48 stored in a memory area 46 in the buffer
21 memory 38, each entry 44 being for storing a checksum value for a corresponding data
22 segment 48 stored in the buffer memory 38. For each data segment 48 retrieved from
23 the disks 29, a checksum value for that data segment is calculated using the checksum
24 circuit 36 (step 52), and an entry 44 in the checksum list 42 is provided for storing the
25 calculated checksum value for that data segment 48 (step 54). The calculated
26 checksum value is then stored in the selected entry 44 in the checksum list 42 (step
27 56). The data segment 48 is stored in the memory 38 (step 58). Preferably,
28 calculating the checksum value for the data segment 48 in step 52 is performed as the

1 data segment 48 is being transferred into the memory 38 from the reach channel 32 in
2 step 58. Data segments 48 and corresponding checksum values can then be
3 transferred out of the memory 38 and provided to the CPU 14 as described above.

4
5 Referring to FIG. 5, in one example, data can be transferred out of the memory
6 38 as packets 60 of data and a checksum value is determined for each packet 60 using
7 the checksum values from the checksum list 42. In packaging the data segments 48
8 into the packets 60 for transfer out of the memory 38, a packet 60 can include a
9 combination of one or more complete data segments 48 and/or one more segment
10 fragments 62. In the following description, in each packet 60, a portion of a data
11 segment 48 which is included in the packet 60 is termed as a data segment fragment
12 62, and the portion of that data segment 48 which is not included in the packet 60 is
13 termed the complementary fragment 64.

14
15 FIG. 5A shows six example packets 60, designated as *packet #0*, *packet #1*,
16 *packet #2*, *packet #3*, *packet #4*, and *packet #5*. Each packet 60 includes at least one
17 complete data segment 48 a data segment fragment 62 of one or more data segments
18 48. Each data segment 48 is 512 bytes in size and each packet 60 is 1460 bytes in
19 size. In FIG. 5A, eighteen data segments 48, indexed as data segment numbers 0
20 through 17, are packaged in the packets *packet #0* through *packet #5* as shown. The
21 *packet #0* contains two complete data segments, numbers 0 and 1, and a fragment 62
22 of a data segment number 2. The *packet #1* contains the complementary fragment 64
23 of the data segment number 2, the complete data segment numbers 3 and 4, and a
24 fragment of the data segment number 5. The *packet #2* contains the complementary
25 fragment of the data segment number 5, the data segment numbers 6, 7 and a
26 fragment of the data segment number 8. The *packet #3* contains the complementary
27 fragment of the data segment number 8, the data segment numbers 9 and 10, and a
28 fragment of the data segment number 11. The *packet #4* contains the complementary

1 fragment of the data segment number 11, the data segment numbers 12, 13, and a
2 fragment of the data segment number 14. And, the *packet #5* contains the
3 complementary fragment of data segment number 14, the data segment numbers 15,
4 16, and a fragment of the data segment number 17.

5
6 As shown in FIG. 5A, for example, when considering *packet #2*, the portion of
7 data segment number 8 included in *packet #2* can be a data segment fragment 62 and
8 the portion of data segment number 8 not included in *packet #2* can be a
9 complementary fragment 64. And, as is inherent in FIG. 5A, when considering *packet*
10 *#3*, the portion of data segment number 8 included in *packet #3* can be a data segment
11 fragment 62 and the portion of data segment number 8 not included in *packet #3* can
12 be a complementary fragment 64.

13
14 Referring to FIG. 6 in conjunction with FIG. 5B, in one example packet building
15 process, one or more complete data segments 48 are packaged in a packet 60 without
16 any data segment fragments (step 70). A checksum value for the packet 60 is
17 determined by retrieving the checksum value for each data segment 48 in that packet
18 60 from a corresponding entry in the check list 42 (step 72); and calculating a
19 checksum value for that packet 60 as a function of the retrieved checksum values (step
20 74). The above steps are applicable to cases where the size of each packet 60 is an
21 integer multiple of the size of each data segment 48. For example, where the size of
22 each data segment 48 is 512 bytes, and the size of each packet 60 is 1536 bytes, each
23 packet 60 can contain three complete data segments 48 therein. In another example, a
24 packet 60 of size 2048 bytes can contain data from four complete 512 byte data
25 segments 48.

26
27 Referring to FIG. 7A in conjunction with FIG. 5A, in another example packet
28 building process, at least one complete data segment 48, and a fragment 62 of one or

1 more data segments are packaged in a packet 60 (e.g., *packet #4*) (step 76). A
2 checksum value for the packet 60 is determined by: retrieving the checksum value for
3 each of said one or more complete data segments 48 in that packet 60 from a
4 corresponding entry in the checksum list 42 (step 78); calculating an intermediate
5 checksum value as a function of the retrieved checksum values (step 80); for each
6 fragment 62 of a data segment 48 in that packet 60, calculating a checksum value for
7 data in that data segment fragment 62 (step 82); and adjusting the intermediate
8 checksum value using the checksum values for said one or more data segment
9 fragments 62 to provide a checksum value for all of the data in that packet 60 (step 84).
10 Step 80 is optional, and packet checksum calculation can be performed in step 84
11 using the checksum values determined in steps 78 and 82.

12
13 Referring to FIG. 7B, in yet another example packet building process, when at
14 least one complete data segment 48, and a fragment 62 of one or more data segments
15 48 are packaged in a packet 60 (e.g., *packet #4*) (step 86), a checksum value for the
16 packet 60 is determined by: retrieving the checksum value for the at least one complete
17 data segment 48 in that packet 60 from a corresponding entry 44 in the checksum list
18 42 (step 88); for each of the one or more data segment fragments 62 in the packet 60:
19 (i) retrieving the checksum value from the entry 44 in the checksum list 42
20 corresponding to the data segment 48 of which that data segment fragment 62 is a part
21 (step 90), and (ii) calculating a checksum value for data in a complementary fragment
22 64 of that data segment 48 (step 92); calculating an intermediate checksum value as a
23 function of the retrieved checksum values (step 94); and adjusting the intermediate
24 checksum value using the checksum values for said one or more complementary data
25 segment fragments 64 to provide a checksum value for all of the data in that packet 60
26 (step 96). Step 94 is optional, and packet checksum calculation can be performed in
27 step 96 using the checksum values retrieved in steps 88, 90, 92.
28

1 Referring to FIG. 7C, in another example packet building process, when at least
2 one complete data segment 48, and a fragment 62 of one or more data segments 48
3 are packaged in a packet 60 (e.g., *packet #4*) (step 98), a checksum value for the
4 packet 60 is determined by: retrieving the checksum value for said at least one
5 complete data segment 48 in that packet 60 from a corresponding entry 44 in the
6 checksum list 42 (step 100); for each of said one or more data segment fragments 48 in
7 the packet 60, if that data segment fragment 62 is less in size than a complementary
8 fragment 64 of the data segment 48 of which that data segment fragment 62 is a part
9 (step 102), then calculating a checksum value for data in that data segment fragment
10 62 (step 104), otherwise, retrieving the checksum value from the entry in the checksum
11 list 42 corresponding to said data segment 48 of which that data segment fragment 62
12 is a part (step 106), and calculating a checksum value for data in said complementary
13 fragment 64 of that data segment (step 108).

14
15 Steps 102 through 108 are repeated as described above for each data segment
16 fragment in the packet. The packet checksum is then determined as a function of the
17 retrieved checksum values and the calculated checksum values (step 110). The step
18 110 of determining the packet checksum can include the steps of determining an
19 intermediate checksum value based on the retrieved checksum values, and adjusting
20 the intermediate checksum value using the calculated checksum values to provide the
21 packet checksum value for all data in the packet 60.

22
23 Referring to FIG. 8 in conjunction with FIG. 5C, in an example packet building
24 process, one or more packets 60 include a fragment 62 of one or more data segments
25 48, without any complete data segments (step 112). A such, each such packet 60
26 includes at least a fragment 62 of a data segment, and no complete data segments. A
27 checksum value for each such packet 60 can be determined by steps including: for
28 each data segment fragment 62 in that packet 60, if that data segment fragment 62 is

1 less in size than a complementary fragment 64 of the data segment 48 of which that
2 data segment fragment 62 is a part (step 114), then calculating a checksum value for
3 data in that data segment fragment 62 by e.g. software instructions (step 116),
4 otherwise, retrieving the checksum value from the entry 44 in the checksum list 42
5 corresponding to said data segment 48 of which that data segment fragment 62 is a
6 part (step 118), and calculating a checksum value for data in said complementary
7 fragment 64 of that data segment (step 120). Steps 114 through 120 are repeated as
8 described above for each data segment fragment 62 in the packet 60. The packet
9 checksum is then determined as a function of the retrieved checksum values and the
10 calculated checksum values (step 122). Alternatively, steps 114, 118 and 120 can be
11 eliminated, and a checksum value directly calculated for each data segment fragment
12 62 according to step 116 regardless of the relative size of the data segment fragments
13 62.
14

15 FIG. 9 shows example steps of another process for packaging (i.e. reblocking)
16 data segments 48 into packets 60, and calculating checksums for the packets 60. The
17 packet building process for each packet 60 comprises a loop starting with initializing a
18 packet checksum (step 124), and determining the next block of data in the area 46 of
19 the buffer memory 38 to include in the packet 60 (step 126). Each block of data can
20 comprise a portion of one or more data segments 48 and/or one or more complete data
21 segments. For this example, each data block can be a complete data segment 48, or a
22 data segment fragment 62. If a next block of data is to be included in the packet 60
23 (step 128), the process determines if a checksum for the data block has been
24 calculated by the checksum circuit 36 (step 130). If so, the process determines if the
25 data block is a complete data segment 48 (step 132). If so, the checksum value for the
26 data segment 48 is retrieved from the checksum table 42 and added to the packet
27 checksum (step 134), and the process proceeds to step 126.
28

1 If in step 130 above, the process determines that a checksum value for the data
2 block has not been determined, then the process calculates a checksum for the data
3 block, wherein the data segment can comprise a complete data segment 48 or a data
4 segment fragment 62 (step 136). That checksum value is then added to the packet
5 checksum value in step 134, and the process proceeds to step 126. If in step 132, the
6 process determines that the data block is not a complete data segment 48, and is a
7 data segment fragment 62, then the process determines if the size of the data segment
8 fragment 62 is less than, or equal to, half the size of a complete data segment 48 of
9 which the data segment fragment 62 is a part (step 138). If so, the process calculates
10 the checksum for that data segment fragment 62 (step 140) and caches the calculated
11 checksum value for the data segment fragment 62 (step 142). The process then adds
12 the calculated checksum value for the data segment fragment 62 to the packet
13 checksum (step 134) and proceeds to step 126.

14
15 If, in step 138, the process determines that the size of the data segment
16 fragment 62 is greater than, or equal to, half the size of the complete data segment 48,
17 then the process calculates the checksum value for the complementary fragment 64 of
18 the data segment fragment 62 (step 144). The process then retrieves the checksum
19 value for the data segment 48 of which the data segment fragment 62 is a part, from
20 the checksum table 42, and determines the checksum for the data segment fragment
21 62 as a function of the retrieved data segment checksum and the calculated
22 complementary fragment checksum (step 146). The process then caches the data
23 segment fragment checksum (step 142), adds it to the packet checksum (step 134) and
24 proceeds to step 126. If in step 128, no other block of data is to be included in the
25 packet, then the packet 60 and its checksum (i.e. the packet checksum) are sent to the
26 packet requester in the network 13, for example (step 148). Steps 124 through 148
27 are repeated for each packet 60 as described above.
28

1 In one example implementation, the present invention can be utilized for
2 calculating checksums in networks using networking protocols such as TCP/IP.
3 Data to be transmitted onto a network typically requires a checksum per packet. The
4 present invention reduces software overhead for determining data segment checksums
5 when data segments 48 are read from e.g. a storage device and reblocked as packets
6 for transmission over the network to a requester. The checksum values for data
7 segments 48 are calculated using the checksum circuit 36 as data segments are
8 transferred from data storage devices such as the disk drive 25 or tape drive into
9 memory such as the buffer memory 38. The calculated checksum values are stored
10 e.g. either in a separate checksum list 42 in the memory 38 as described above, or as a
11 new field in the sector command descriptors in the disk control block of a disk drive
12 ASIC. In one version, checksum values are calculated for every disk segment 48,
13 wherein each disk data segment 48 is 512 bytes in size. Checksum values can also be
14 calculated for sub- data segment blocks of data.

15
16 Data packets 60 are composed from combinations of whole (complete) and/or
17 partial (fragment) data segments 48. The data segment checksum values are then
18 used to determine packet checksums for outgoing network data packets 60 via addition,
19 corrected for data segment fragments 62 used in some data packets 60. The packet
20 checksum is then added to the network header checksum as appropriate, and stored
21 into the packet header. Although the example herein is for a disk drive 25, the present
22 invention is equally applicable to a tape drive or a server computer with a checksum-
23 capable host interface. The checksumming technique of the present invention can also
24 be applied where the data is to be encrypted before transmission. Because the
25 checksum for the data segments 48 is calculated using a checksum circuit 36, rather
26 than by software instructions executed by a processor, as a result CPU and memory
27 bandwidth utilization is reduced, leading to improved throughput and latency in network-
28 attached disk and tape drives.

1 In network environments utilizing the TCP/IP protocol suite, all data packets 60
2 using TCP, and many using UDP, as the transport protocol contain a data payload
3 checksum. The data payload is comprises combinations of complete (whole) data
4 segments 48, and/or fragments (portions) 62 or 64 of data segment 48. Since the
5 payload checksum is stored in a packet header, the payload checksum must be
6 calculated before a packet 60 is sent to the network. Conventional implementations of
7 TCP/IP calculate the payload checksum in software as a separate pass over the
8 payload data, thereby consuming precious CPU cycles and memory bandwidth. In
9 several implementations of TCP/IP, the payload checksum is calculated as part of a
10 data copy operation. The data is typically copied from a user buffer memory to a
11 system buffer memory before transmission onto the network. Utilizing pipelined
12 microprocessor architectures, the CPU instructions (e.g. addition) required to calculate
13 the packet checksum is executed as the system microprocessor loads data from buffer
14 memory for copying. This is termed "folding in" the checksum with the data copy. Other
15 conventional TCP/IP implementations use "zero copy" techniques, so that payload data
16 is not copied from one location to another location as part of packet transmission.
17 Since the payload data is not copied, folding in the checksum with the data copy is
18 inapplicable.

19
20 The present invention can be applied to improve all said conventional
21 implementations, whereby data segment checksum values are calculated using the
22 checksum circuit 36 according to the present invention, and the packet checksum is
23 determined as a function of the data segment checksum values. Specifically, the
24 transport-level checksum value in TCP packets, and optionally in UDP packets, is a
25 simple 16-bit ones-complement addition value. The checksum value stored in the
26 packet 60 is the ones-complement of the addition value, so that on checksumming the
27 packet with the checksum, the result is zero. This checksum allows end-to-end
28 checking of data, while link-layer cyclic redundancy checks (CRCs) can be used to

1 detect on-the-wire corruption. The TCP checksum protects against most software-
2 caused inadvertent packet corruption, e.g. in routers. FIG. 10A illustrates an example
3 layout for a TCP/IP packet 150 in an Ethernet frame, generally indicating the portions of
4 the packet 150 covered by the relevant checksums and CRC values. The TCP
5 checksum covers a data payload 152 and most, but not all, of IP and TCP headers 154
6 and 156, respectively. FIG. 10B illustrates an example IP header 154 with normal
7 fields. In particular, the time-to-live (TTL) Offset value and the header checksum fields
8 of the IP header 154 are not checksummed. Any IP options are also not covered.
9 Referring to FIG. 10C, a pseudo header 158 is generated from the IP header 154 and is
10 provided to TCP/IP for checksum.

11
12 An important benefit of using checksums is the ease with which sub-checksums
13 for portions of data in a packet 150 can be calculated to create a complete checksum
14 for at least all the data payload in a packet 150. The checksum can be easily corrected
15 for excess data. Referring back to FIGS. 1, 3, 5A-C and 10, in an example, the disk
16 drive 25 utilizes TCP/IP in an Ethernet environment, with 1500 byte network packets
17 150, each composed of 1460 byte data payload 152 and a 40 byte TCP/IP header 160
18 which includes the IP and TCP headers 154, 156, respectively. As each 512-byte data
19 segment 48 is transferred from the disk 29 into the buffer memory 38, a checksum
20 value for the data segment 48 is calculated by the checksum circuit 36, and stored in
21 the checksum list 42. Then for transferring data segments 48 out of the buffer memory
22 38, combinations of one or two complete data segments 48 and/or one or two data
23 segment fragments 62 are used to compose a data payload 152 for each TCP packet
24 150, based on the alignment of the TCP packet 150 relative to data segment
25 boundaries.

26
27 Where only complete data segments 48 are included in a packet 150, then the
28 checksum values for the complete data segments 48 are retrieved from the checksum

1 list 42 and added together to generate the packet checksum value. Where fragments
2 62 of data segments 48 are also included in the packet 150, for each included segment
3 fragment 62, the smaller of that segment fragment 62 and the excluded complementary
4 fragment 64, is checksummed in software and is used to generate the packet
5 checksum. Specifically, if the size of the included segment fragment 62 is less than, or
6 equal to, half the size of the data segment 48, then the checksum for the segment
7 fragment 62 is calculated in software, and added to the retrieved checksums for the
8 complete data segments in the packet 150 to generate the packet checksum.

9
10 Otherwise, if the included segment fragment 62 is more than half the size of the
11 data segment 48, then: (i) the checksum for that data segment 48 is retrieved from the
12 checksum list 42, (ii) the checksum for the excluded complementary fragment 64 of that
13 data segment 48 is calculated in software and subtracted from the retrieved checksum
14 for that data segment 48, and (iii) the difference is added to the retrieved checksums for
15 complete data segments 48 to generate the total packet checksum. In either case, the
16 checksums calculated in software are cached in a memory area, such as in memory 38
17 or memory 16, so that they may be reused in a similar checksum process for the next
18 TCP packet 150 which will likely include said excluded complementary fragments 64.

19
20 The placement of data segment boundaries relative to packet boundaries can be
21 a uniform random distribution, and checksum values for consecutive packets 150 are
22 determined as described above. In the above example, because checksums are
23 calculated in software only for a smaller fragment 62 of a data segment 48, the amount
24 of data checksummed in software is always less than 256 bytes. Therefore, with said
25 random placement, the average amount of data checksummed in software in each
26 packet 150 is 128 bytes, compared to 1460 bytes in conventional methods where the
27 checksum for all payload data in each packet 150 is calculated in software.

Referring back to the packets in FIGS. 5A-C, example alignment of the packets 60 relative to the boundaries 177 of data segment 48 are shown. Further, data segment fragments 62 and complementary fragments 64 that are checksummed in software are highlighted in gray. The table in FIG. 11 shows data segment offsets for the packets shown in FIG. 5A, where *packet #0* starts with a data segment offset of zero. A fragment 62 of a data segment within a current packet 60 under consideration is referred to as the “inner” fragment, and the unused complementary fragment 64 outside the current packet 60 is referred to as the “outer” fragment. Further, a data segment fragment 62 at the front of the packet 60, from left to right, is referred to as the “leading” fragment, and a data segment fragment at the end of the packet 60 is referred to as the “trailing” fragment. As such, the inner trailing data segment fragment 62 for one packet becomes the outer leading fragment 64 for the next packet. In FIG. 5A, all packets after *packet #0* obtain the checksum for leading data segment fragments therein from the checksum values cached for previous packets as described above. The software checksum size in said table 42 is for the trailing fragments in each packet.

FIG. 12 shows a block diagram of the architecture of another embodiment of the data controller 30 (FIG. 2) including an example checksum circuit 36, receiving data from data disks 29 through the read channel 32. The data controller 30 includes a sequencer 162, the buffer memory 38 (e.g., a cache buffer), a buffer controller 164, the checksum circuit 36, a ROM 166, and the data path microcontroller 168, interconnected to a microbus control structure 170 and the buffer data bus 33 as shown. The data controller 30 further includes a high level interface controller 172 implementing a bus level interface structure, such as SCSI II target, for communications over the bus 22 with the I/O Adaptor 18 (e.g. SCSI II host initiator adapter) within the computer system 10 (FIG. 1).

1 The microcontroller 168 comprises a programmed digital microcontroller for
2 controlling data formatting and data transfer operations of the sequencer 162 and data
3 block transfer activities of the interface 178. The channel 32 can utilize a "partial
4 response, maximum likelihood detection" or "PRML" signaling technique. PRML
5 synchronous data detection channels frequently employ analog-to-digital converters
6 such as e.g. six-bit flash analog-to-digital converters for providing synchronous samples
7 of the analog signal read from the data disks 29. One example of a PRML synchronous
8 data detection channel is found in commonly assigned U.S. Pat. No. 5,341,249,
9 entitled: "Disk Drive Using PRML Class IV Sampling Data Detection with Digital
10 Adaptive Equalization", the disclosure thereof being incorporated herein by reference.

11
12 The drive electronics 25 includes a preamplifier circuit 174 and voltage gain
13 amplifier ("VGA") 176, such that magnetic flux transitions sensed by the selected data
14 transducers 27 are preamplified as an analog signal stream by the preamplifier circuit
15 174, and passed through the VGA 176 for controlled amplification on route to the
16 channel 32. The channel 32 can comprise an analog programmable filter/equalizer
17 178, a flash analog-to-digital ("A/D") converter 180, a digital adaptive finite impulse
18 response ("FIR") filter 182, a Viterbi detector 184, and a postcoder 186. The voltage
19 controlled analog signal stream from the VGA 176 is passed through the programmable
20 analog filter/equalizer 178. Equalized analog read signals from the filter 178 are then
21 subjected to sampling and quantization within the A/D converter 180. Data samples
22 from the A/D converter 180 are then passed through the FIR filter 182 which employs
23 adaptive filter coefficients for filtering and conditioning the raw data samples in
24 accordance e.g. with desired PR4 channel response characteristics in order to produce
25 filtered and conditioned data samples. The bandpass filtered and conditioned data
26 samples leaving the FIR filter 182 are then passed to the Viterbi detector 184 which
27 decodes the data stream, based upon the Viterbi maximum likelihood algorithm
28 employing a lattice pipeline structure implementing a trellis state decoder, for example.

1 The decoded data put out by the detector 184 are passed through a postcoder 186
2 which restores the original binary data values.

3
4 The binary data values are deserialized by an encoder-decoder/serializer-
5 deserializer 188 ("ENDEC/SERDES"), wherein the ENDEC/SERDES 188 frames and
6 puts out e.g. eight bit user bytes in accordance with an inverse of the 8/9ths rate coding
7 convention (nine code bits for every eight user data bits). The ENDEC/SERDES 188
8 can be in accordance with commonly assigned U.S. Pat. No. 5,260,703, the disclosure
9 of which is incorporated herein by reference. The decoded user bytes are then passed
10 to the sequencer 162 along the path 190, and the sequencer 162 stores the user data
11 bytes as data segments 48 in the memory 38 via the bus 33. The sequencer 162
12 sequences the data segments 48 between the buffer memory 38 and data disks 29.
13 The buffer memory controller 164 controls the buffer memory 38. The bus level
14 interface circuit 172 transfers data segments between the buffer memory 38 and the
15 bus 22. The microcontroller 168 controls the sequencer 162, buffer controller 164 and
16 the bus level interface circuit 172. The microcontroller 168 can also include functions
17 for transducer positioning servo loop control via the servo controller 34 (FIG. 2).

18
19 Specifically, the data sequencer 162 sequences blocks of data (e.g. data
20 segments 48) to and from the disks 29 at defined data block storage locations thereof.
21 The sequencer 162 is connected to the read/write channel 32 and is directly responsive
22 to block segment counts for locating and assembling in real time the data segments 48
23 read from and written to the data storage surfaces of disks 29, and for handling data
24 segment transfers between the disks 29 and the buffer memory 38. The buffer memory
25 controller 164 generates and puts out addresses to the buffer memory 38 for enabling
26 the buffer memory 38 to transfer data segments 48 to and from the sequencer 162 via
27 the bus 33. The microcontroller 168 includes data block transfer supervision routines
28 for supervising operations of the sequencer 162, the buffer memory controller 164 and

1 the host bus interface 172. The microcontroller 168 can further include servo
2 supervision routines for controlling the servo controller 34 for positioning the
3 transducers 27 over data tracks on disks 29 for read/write operations. The commonly
4 assigned U.S. Pat. No. 5,255,136, entitled "High capacity submicro-winchester fixed
5 disk drive", issued to Machado, et. al, provides a detailed example of the operation of
6 the sequencer 162 in conjunction with the reach channel 32. The disclosure of the U.S.
7 Pat. No. 5,255,136 is incorporated herein by reference.

8
9 As data segments 48 are transferred by the sequencer 162 into the buffer
10 memory 38, the checksum circuit 36 calculates checksum values for the data segments
11 48 and stores the checksum values in the checksum table 42. The checksum circuit 36
12 can comprise a logic circuit configured to perform a checksum function. For example, a
13 logic circuit for the above TCP example can comprise a ones-complement adder. The
14 checksum circuit 36 can also comprise a PLD or programmable gate array, configured
15 with a set of instructions to create a logic circuit for checksumming. Examples of such
16 program instructions can be found in "Implementing the Internet Checksum in
17 Hardware", by J. Touch and B. Parham, ISI, RFC 1936, April 1996; and "Computing the
18 Internet Checksum", by R. Braden, D. Borman and C. Partridge, ISI, RFC 1071,
19 September 1988, incorporated herein by reference.

20
21 The checksum circuit 36 automatically calculates a checksum value for each
22 data segment 48 read from the disks 29 and stores the checksum value in the
23 checksum table 42. In one example, the microcontroller 168 allocates the checksum
24 table 42 in the memory 38, and provides the checksum circuit 36 with the base address
25 of the table 42 in the buffer memory 38. Then for each incoming data segment 48, the
26 checksum circuit 36 calculates a checksum value, and also shifts down the address in
27 buffer memory 38 where the data segment 48 is being stored (e.g. shift down towards
28 the least significant bits by 9 bits). The checksum circuit 36 then uses the shifted down

1 value as an index to the checksum table 42 for storing the checksum value for the data
2 segment 48. Specifically, utilizing the base address of the checksum table 42 and the
3 index, the checksum circuit 36 determines the entry location within the table 42 to store
4 the checksum value calculated for the data segment 48. The checksum circuit 36 then
5 repeats the process for the next incoming data segment 48 as transferred from the
6 sequencer 162 to the memory 38 via the bus 33.

7
8 FIG. 13 shows a block diagram of the architecture of an embodiment of the
9 checksum circuit 36 of FIG. 12. The checksum circuit 36 comprises an adder 191; an
10 accumulator ("ACC") 192; a one's complement adder ("OCA") 194 having the same bit
11 width as the buffer data bus 33; a buffer start address register ("BSA") 196; a
12 decrementing counter 198; a buffer count register ("BCR") 200; a barrel shifter ("BS")
13 202; a buffer size register ("BSR") 204; checksum base address register ("CBA") 206
14 for checksum table 42 in buffer RAM 38; a buffer data bus interface ("BIU") 208 to
15 support passive listening, arbitration and data driving capabilities of the checksum
16 circuit 36; a read/write checksum control register ("CCR") 210 with enable/disable (W)
17 and sequencing error (R); and a checksum microsequencer ("CSMS") 212.

18
19 In operation, the checksum circuit 36 is initialized by the microcontroller 168 at
20 the start. The microcontroller 168 allocates an area of the buffer RAM 38 for the
21 checksum table 42, wherein the size of the checksum table 42 can be e.g. the product
22 of the number of data segments 48 and the size of a checksum table entry, such as
23 eight, sixteen or thirty two bits per entry. The microcontroller 168 then writes the base
24 address of the table 42 into the CBA register 206. As such, the CBA register 206 is not
25 modified by the checksum circuit 26. The CBA register 206 is a configuration register
26 written by the microcontroller 186 when the checksum circuit 36 is initialized. The
27 microcontroller 168 then enables the checksum circuit 36 by writing an enable bit in the
28 checksum control register CCR 210.

1 The sequencer 162 writes the memory address of the data buffer 46 for string
2 data segments 48 in the memory 38 into the BSA register 196 before transferring data
3 onto the buffer data bus 33 to be stored in the memory 38. The microcontroller 168 can
4 write the size of the data buffer 46 in the BSR register 204. The CSMS 212 copies the
5 contents of the BSA register 168 into the BCR register 200 at the start of a segment
6 write to the memory 38. The microsequencer CSMS 212 checks the buffer count
7 register BCR 200, and if the count therein is non-zero, the CSMS 212 signals an error
8 to the microcontroller 168. The error signal indicates that the checksum circuit 36 is
9 currently computing a checksum value for another data segment, and is being asked to
10 start another checksum calculation. This situation can arise when a sequencing or
11 hardware error results in fewer bytes than the size of a data segment to be included in a
12 currently active checksum calculation. Such errors can include incorrect setting of the
13 BSR register 204, or the BIU 208 missing one or more words of data segments 48
14 written onto the data bus 33.

15
16 If the count in the BCR 200 is zero, the CSMS 212 writes "0" into the
17 accumulator ACC 192, copies the contents of the register BSR 204 into the BCR
18 register 200, and starts passively listening on buffer data bus 33 via the BIU 208 for
19 data transfer from the sequencer 162 to the buffer RAM 38. As data is written by the
20 sequencer 162 into buffer RAM 38 via the buffer data bus 33, the BIU 208 listens to the
21 data on the buffer data bus 33, and the CSMS 212 passes a copy of each word of the
22 data on the bus 33 to the ones-complement adder OCA 194, which adds it to the
23 accumulator ACC 192 and stores the value back in the ACC 192. The CSMS 212 then
24 instructs the decrementing counter 198 to subtract one from the BCR 200. When the
25 BCR 200 reaches zero, the CSMS 212 calculates the address in memory to store the
26 checksum accumulated in ACC 192.

1 Said address for storing the accumulated checksum is calculated by the adder
2 191 as follows. The start address of the data buffer 46, stored in the BSA register 196,
3 is shifted down towards its least significant bits by the shifter BS 202. The address in
4 the BSA register 196 is shifted down by a number of bits represented by the difference
5 ($\log_2(\text{BSR}) - \log_2(\text{BWS})$), wherein BWS is the bus word size in bytes. For example, if a
6 data segment 48 is 512 bytes in size, and the bus 33 is thirty two bits wide (i.e. four
7 bytes), the shift down value is: ($\log_2 512 - \log_2 4$) = 9 - 2 = 7. In this example, the
8 accumulator ACC 192 has the same width as the data bus 33. The BSA register 196 is
9 shifted down in the BS 202 by seven bits to provide the offset from the checksum table
10 42 base address in the CBA register 206. The offset from the BS 202, and the
11 checksum table 42 base address in the CBA register 206 are input to the adder 191 to
12 calculate the address to store the data segment checksum. The adder 191 adds the
13 contents of the CBA register 206 to the shifted start address from the BS 202 (offset) to
14 calculate said address for storing the checksum accumulated in the ACC 192. The
15 checksum accumulated in the accumulator ACC 192 is then written to the calculated
16 address in the checksum table 42. Other embodiments of the checksum circuit 36 are
17 possible and contemplated by the present invention. Further the steps performed by
18 the microcontroller 168 can be performed by the CPU 14 instead, or in conjunction with
19 the microcontroller 168.

21 In the above embodiments of the present invention, the checksum circuit 36 is
22 integrated into the data controller 30 for calculating checksums for data transferred
23 from the disk 29 into the buffer memory 38 in the data controller 30. The memory 38
24 can be external to the data controller 30 within the drive electronics 26. Alternatively,
25 the checksum circuit 36 can be incorporated into the read channel 32 of the disk drive
26 or tape drive electronics ASIC, wherein the buffer memory 38 is connected to the read
27 channel 32. After determining a checksum value for every 512 by data segment 48
28 from the disks 29, the checksum value is stored into the checksum table 42 in the

1 memory 38. In either case, each data segment 48 is checksummed as it is transferred
2 into the memory 38, and the checksum value for that data segment 48 is stored into a
3 entry 44 in the checksum table 42 upon completion of the data segment transfer. As
4 detailed above, in one embodiment of the checksum circuit 36, a base register defining
5 the start address of the table 42 is utilized and the checksum value for each data
6 segment 48 is stored into table 42 with the same index as in the memory area 46 into
7 which the data segment 48 is stored. In another example, the checksum value for each
8 data segment is stored as part of the data segment transfer control block on disks 29,
9 along with such information as the segment status.

10
11 Referring to FIG. 14, in another embodiment of the present invention, the I/O
12 adapter 18 includes the checksum circuit 36 and the buffer memory 38. The buffer
13 memory 38 can be part of the main system memory 16. The I/O adapter 18 can
14 comprise e.g. an ATA or SCSI interface card. Data segments 48 from the disk drive 25
15 are transferred to the I/O adapter 18 via the bus 22, wherein the I/O adapter 18
16 calculates checksums for data segments 48 using the checksum circuit 36 as they are
17 received therein via the bus 22. The checksums are stored in the checksum list 42 as
18 described above. In paged memory systems, the checksum for each page can also be
19 stored in the same area of kernel memory that is used to control the page.
20

21 Further, for host interface checksums, the storage device 25 can calculate the
22 checksums rather than the host bus adapter 18. The communication of said
23 checksums to the host (e.g. CPU 14) is then accomplished by a MESSAGE IN phase
24 on a SCSI bus. Specifically, after the storage device finishes a DATA IN phase in
25 which data is transferred to the host, the storage device then changes to a MESSAGE
26 IN phase and transfers the checksums corresponding to the transferred data.
27
28

1 The present invention can be used in any data processing environment where
2 units of data such as blocks, segments, sectors, etc. are buffered for transmission onto
3 a network such as a network system 19 (FIG. 1). Typical applications include e.g.
4 Network File Service (NFS) and Common Internet File Service (CIFS) bulk data service
5 protocols. The data buffering can be performed within a storage device such as the
6 disk drive 25. Further, the disk drive 25 can provide an IP-based bulk data service.
7 Alternatively, the data buffering can be performed within a file server computer such as
8 a Network Appliance or a Unix or NT machine operating as a file server, where the
9 checksum is made available through a storage device interface. Further, the present
10 invention can be used in any networked data processing system such as a system
11 utilizing TCP/IP. In the latter case, when the TCP data payload size is an integer
12 multiple of the data segment size, management of data segment fragment checksums
13 may not be required.
14

15 Accordingly, referring back to FIG. 1, in another aspect of the present invention,
16 the computer system 10 can further comprise a network interface device 15 connected
17 to the bus 20 for data communication between the computer system 10 and other
18 computer systems 11 via a network link 13 in the networked data processing system
19 19. The checksum circuit 36 can be placed anywhere along the data path from the
20 data disks 29 in the disk drive 25 and the network link 13 connected to the network
21 interface device 15.
22

23 As such, the checksum circuit 36 can be moved and placed in the final stage of
24 moving retrieved data segments 48 to the network interface device 15. For example, as
25 shown in FIG. 15, in one embodiment of the present invention, the network interface
26 device 15 includes the checksum circuit 36 for calculating and storing checksum values
27 for the data segments 48 in the checksum list 42. The buffer memory 38 can be e.g. a
28 part of the network interface device 15, or a part the main system memory 16. Data

1 segments 48 are retrieved from the disk drive 25 and transferred to the network
2 interface device 15 without calculating checksums. Specifically, data segments 48 from
3 the disk drive 25 are transferred to the I/O adapter 18 via the I/O bus 22, then to the
4 system bus 20 via the I/O adapter 18, and then to the network interface 15 via the
5 system bus 20. The data segments 48 stored in the buffer memory 38, and the data
6 segments 48 can be modified before inclusion in packets 150 and transmission over the
7 network link 13. The network interface device 15 can include circuits and routines for
8 performing steps of determining the parts of each packet 150 that are to be
9 checksummed and the possibly variable location of the checksum's insertion point
10 offset. For example, Ethernet headers, the IP TTL, IP fragment headers and IP
H options, and trailing Ethernet padding are not checksummed. When the checksum
B engine 36 is integrated in the network interface device 15, specific routines can be
I utilized to provide data encryption (e.g., IPSEC payload encryption) after the TCP
13 packet checksum calculation wherein the checksum itself is encrypted.
14

15
16 In the embodiments of the present invention described herein, the reblocking
17 processes described above in conjunction with FIGS. 4, and 6-9, can be used to
18 configure the CPU 14 or the microcontroller 168 to perform said reblocking processes.
19 In the latter case, program instructions (software) embodying said reblocking processes
20 can be stored in the memory 16 and executed by the CPU 14 (FIG. 1). The CPU 14
21 retrieves the checksum values from the checksum table 42 within the memory 38, and
22 calculates the packet checksums as described above. Further, the checksum values
23 for data segment fragments 62 and complementary fragments 64 are computed utilizing
24 computer instructions executed by the CPU 14.
25

26 Appendix A provides example program instructions for computing a partial
27 checksum TCP/UDP fragments as described above. Further, Appendix B provides
28 example program instructions for reblocking data segments into e.g. TCP/IP packets

1 150, and calculating checksums for packets 150 according to a reblocking process such
2 as described above in conjunction with the flow diagram of FIG. 9.

3
4 In on example checksumming and reblocking operation wherein the CPU 14
5 executes the reblocking program instructions, data segments 48 flow from the disks 29
6 through the read channel 32 to the buffer memory 38, wherein the checksum circuit 36
7 calculates a checksum value for each data segment. The checksum circuit 36 can
8 reside e.g. in the channel 32, in the data controller 30, in the I/O adapter 18 or in the
9 network interface 15, for example. The memory area 46 or the buffer memory 38 for
10 buffering the data segments can be e.g. in the memory 16, in the electronics 26, in the
11 I/O adapter 18, or in the network interface 15. The checksum table 42 can be e.g. in a
12 designated area in the memory 38 or in the memory 16. The CPU 14 can be primarily
13 responsible for network protocol program instructions. The reblocking software in the
14 CPU 14 is invoked to form packets 150 of data segments 48 to transfer from the
15 memory 38 through the network 19 via the network link 13. Routines according to
16 Network protocol stacks such as TCP/IP, can be included in the CPU 14 to recognize
17 the completion of data segments 48 arriving from disks 29 into the buffer memory 38,
18 and then to execute the reblocking process to create the packets 150, and other
19 protocol processing code, before the packets 150 are sent out through the link 13 via
20 the network interface 15.

21
22 As configured by the reblocking program instructions, the CPU 14 calculates the
23 memory address and length of each data block for inclusion in a data packet 150. As
24 described above in conjunction with TCP/IP implementations of the process of the
25 present invention, if the memory address of a data block is at the boundary 177 of a
26 data segment 48 (FIG. 5A), then to retrieve the checksum value for that data segment
27 48, the CPU 14 divides the memory address (e.g. 32 bits) of the data segment 48 by
28 the size of the data segment e.g. 512. The division is equivalent to discarding the nine

1 least significant bits of the memory address (or shifting the high order twenty three bits
2 of the address down by nine), and using the shifted value as an index into the
3 checksum table 42 to retrieve the checksum of that data segment 48. The checksum
4 table 42 is indexed by data segment number, which is determined by the memory
5 address of the data segments 48 in the buffer memory 38.

6
7 If the memory address of the data block (i.e. start address of a data block to be
8 included in a packet 150) is within the data segment 48, then to determine the byte
9 offset of said data block from the beginning of the data segment 48, the CPU 14 utilizes
10 the nine least significant bits of the data block memory address to determine said
11 offset. The size of a data segment fragment, defined by the start address of the data
12 block within the data segment 48 and a boundary of the data segment 48, is also
13 determined. The data segment fragment for inclusion in the packet is checksummed as
14 detailed above. If the length of the data block is such that the data block is entirely
15 within the boundaries of a data segment, then in one example, the data in the data
16 block can be checksummed completely in software. Alternatively the checksum value
17 for the data block can be determined as a function of one or more of the checksum
18 values in the checksum table 42 and/or cached checksum values as described above.

19
20 For example, referring back to FIG. 5A, if the memory address of the first data
21 block to be stored in the data packet #1 is 1460, then dividing 1460 by 512 as the size
22 of the data segments 48 provides an index value of 2 into the checksum table 42 for
23 retrieving the checksum value of a corresponding target data segment (e.g. data
24 segment number 2). Further, the least significant nine bits of that memory address
25 provide the offset value of 436 bytes within said target data segment. As such the data
26 block to be included in the packet begins at an offset of 436 bytes from the beginning of
27 the target data segment in the buffer memory 38. Because 436 bytes is more than half
28 of the data segment length of 512 bytes, the CPU 14 performs a software checksum for

1 the 76 byte fragment 64 of the data segment 48 as a first data portion of *packet #1* in
2 FIG. 5A shown in gray.

3
4 Referring to FIG. 12, in another embodiment of the present invention where the
5 microcontroller 168 executes reblocking process, the microcontroller 168 retrieves the
6 checksum values from the checksum table 42 within the memory 38, and calculates the
7 packet checksums as described above. Further, the checksum values for data
8 segment fragments 62 and complementary fragments 64 are computed utilizing
9 computer instructions executed by the microcontroller 168. The example program
10 instructions in Appendix A and Appendix B are used to configure the microcontroller
11 168 to perform the reblocking processes. The microcontroller 168 and related circuitry
12 such as the bus 170, buffer controller 164, ROM 166, buffer RAM 38 and the checksum
13 circuit 36, can be placed along the data path from the disks 29 to the host interface 15
14 as described above.

15
16 In on example checksumming and reblocking operation, data segments 48 flow
17 from the disks 29 through the read channel 32 to the buffer memory 38, wherein the
18 checksum circuit 36 calculates a checksum value for each data segment. The
19 reblocking software in the microcontroller 168 is invoked to form packets 150 of data
20 segments 48 to transfer from the memory 38 to the host interface 172. Routines
21 according to Network protocol stacks such as TCP/IP, can be included in the
22 microcontroller 168 to recognize the completion of data segments 48 arriving from disks
23 29 into the buffer memory 38, and then to execute the reblocking process to create the
24 packets 150, and other protocol processing code, before the packets 150 are sent to
25 the interface 172.

26
27 As configured by the reblocking program instructions, the microcontroller 168
28 calculates the memory address and length of each data block for inclusion in a data

1 packet 150. As described above in conjunction with TCP/IP implementations of the
2 process of the present invention, if the memory address of a data block is at the
3 boundary 177 of a data segment 48 (FIG. 5A), then to retrieve the checksum value for
4 that data segment 48, the microcontroller 168 divides the memory address (e.g. 32 bits)
5 of the data segment 48 by the size of the data segment e.g. 512. The division is
6 equivalent to discarding the nine least significant bits of the memory address (or shifting
7 the high order twenty three bits of the address down by nine), and using the shifted
8 value as an index into the checksum table 42 to retrieve the checksum of that data
9 segment 48. The checksum table 42 is indexed by data segment number, which is
10 determined by the memory address of the data segments 48 in the buffer memory 38.

11
12 If the memory address of the data block (i.e. start address of a data block to be
13 included in a packet 150) is within the data segment 48, then to determine the byte
14 offset of said data block from the beginning of the data segment 48, the microcontroller
15 168 utilizes the nine least significant bits of the data block memory address to
16 determine said offset. The size of a data segment fragment, defined by the start
17 address of the data block within the data segment 48 and a boundary of the data
18 segment 48, is also determined. The data segment fragment for inclusion in the packet
19 is checksummed as detailed above. If the length of the data block is such that the data
20 block is entirely within the boundaries of a data segment, then in one example, the data
21 in the data block can be checksummed completely in software. Alternatively the
22 checksum value for the data block can be determined as a function of one or more of
23 the checksum values in the checksum table 42 and/or cached checksum values as
24 described above.

25
26 For example, referring back to FIG. 5A, if the memory address of the first data
27 block to be stored in the data *packet #1* is 1460, then dividing address 1460 by 512 as
28 the size of the data segments 48 provides an index value of 2 into the checksum table

1 42 for retrieving the checksum value of a corresponding target data segment (e.g. data
2 segment number 2). Further, the least significant nine bits of that memory address
3 provide the offset value of 436 bytes within said target data segment. As such the data
4 block to be included in the packet begins at an offset of 436 bytes from the beginning of
5 the target data segment in the buffer memory 38. Because 436 bytes is more than half
6 of the data segment length of 512 bytes, the microcontroller 168 performs a software
7 checksum for the 76 byte fragment 64 of the data segment 48 as a first data portion of
8 *packet #1* in FIG. 5A shown in gray.

9
10 In another embodiment, the reblocking program instructions can be packaged
11 into two groups of modules such that the CPU 14 executes a first group of the modules
12 and the microcontroller 168 executes a second group of the modules, whereby the
13 microcontroller 168 operates in conjunction with the CPU 14 to accomplish the
14 reblocking steps above.

15
16 In one aspect, the present invention provides significant performance
17 improvements for data service from disk or tape drives through a buffer memory onto a
18 network. Calculating checksums using a checksum circuit 36 as data is transferred into
19 a buffer memory 38 substantially reduces memory bandwidth consumed and the
20 number of CPU instruction cycles needed for software checksums. Further, data
21 transfer latency for checksums is reduced. For example, as is conventional, a software
22 checksum routine coded in x86 assembler language uses requires execution of
23 nineteen CPU instructions to checksum every thirty two bytes of data, requiring
24 execution of about 870 instructions to checksum 1460 bytes of data. By contrast,
25 utilizing the present invention, checksumming 1460 bytes of data requires on average
26 execution of only eight CPU instructions, providing a reduction of 790 instruction
27 executions per packet.

1 A conventional software TCP/IP implementation in a general-purpose operating
2 system requires about 4,000 CPU instructions for software checksum calculation per
3 packet, including one data copy and one checksum pass. In such conventional
4 implementations, more than 20% of the CPU instruction executions per packet sent are
5 for checksum calculation. By utilizing the present invention for calculating checksums,
6 a reduction of 790 instruction executions per packet is achieved, effectively providing
7 about a 20% reduction in the CPU cost of TCP/IP transmissions. Further, in a
8 conventional one-copy TCP/IP stack, data moves across the memory bus four times --
9 once on data read before being transmitted onto the network, once for software
10 checksum calculation, and twice for data copy (e.g., read and write). Utilizing the
11 present invention for calculating checksum values reduces the memory bandwidth
12 requirements by 25%.
13

14 Where a conventional zero-copy TCP/IP stack is used in an embedded, non-
15 virtual memory software environment rather than a general purpose operating system,
16 the total overhead for packet transmission can be between 2000 to 3000 instructions
17 per packet. Utilizing the present invention eliminates 790 instructions per packet sent
18 and can reduce CPU utilization by about 30%. Where a conventional zero-copy TCP
19 stack is used, the software checksum represents half of the total memory bandwidth
20 requirements. Utilizing the present invention, the memory bandwidth requirements for
21 checksum calculation can be reduced by about 50%.
22

23 As described herein, a checksum value calculated for a data segment 48 and
24 stored in the checksum list 42 need not be the actual checksum for the data segment
25 48. The checksum value can be e.g. a quantity which is a function of the actual
26 checksum. In the example program instructions in Appendix B, the checksum value is
27 a 32-bit number representing the actual checksum, wherein the 32-bit number
28 checksum value can be converted to the actual checksum by adding the two 16-bit

1 halves of the 32-bit number checksum value together, and performing a logical NOT on
2 the result to obtain an actual 16-bit checksum transmitted in the packet. The same
3 technique applies to the checksum values calculated for the data segment fragments
4 and the checksum values for the complementary fragments.

5
6 The present invention has been described in considerable detail with reference
7 to certain preferred versions thereof; however, other versions are possible. Therefore,
8 the spirit and scope of the appended claims should not be limited to the description of
9 the preferred versions contained herein.

What is claimed is:

1. A method of providing checksum values for data segments retrieved from a data storage device for transfer into a buffer memory, comprising the steps of:

(a) maintaining a checksum list comprising a plurality of entries corresponding to the data segments stored in the buffer memory, each entry for storing a checksum value for a corresponding data segment stored in the buffer memory; and

(b) for each data segment retrieved from the storage device:

(1) calculating a checksum value for that data segment using a checksum circuit for performing a checksum function;

(2) storing the checksum value in an entry in the checksum list;

and

(3) storing that data segment in the buffer memory;

wherein, for transferring packets of data out of the buffer memory, a checksum value can be calculated for data in each packet based on one or more checksum values stored in the checksum list.

2. The method of claim 1, wherein step (b)(1) further includes the steps of calculating the checksum for said data segment using the checksum circuit as the data segment is transferred into the buffer memory.

3. The method of claim 1 further comprising the steps of:

(c) building packets of data for transfer out of the buffer memory, and

(d) providing a checksum value for data in each packet based on one or more checksum values stored in the checksum list.

4. The method of claim 3, wherein:

each packet in a set of said packets includes one or more complete data segments; and

1 the step (d) of providing a checksum value for each of the packets in said
2 set of packets further includes the steps of:

3 (1) retrieving the checksum value for each data segment in that
4 packet from a corresponding entry in the checksum list; and

5 (2) calculating a checksum value for that packet as a function of
6 the retrieved checksum values.

7
8 5. The method of claim 3, wherein:

9 each packet in a set of said packets includes: (i) one or more complete
10 data segments, and (ii) a fragment of each of one or more data segments; and

11 the step (d) for providing a checksum value for each of the packets in said
12 set of packets further includes the steps of:

13 (1) retrieving the checksum value for each of said one or more
14 complete data segments in that packet from a corresponding entry in the checksum list;

15 (2) for each data segment fragment in that packet, calculating a
16 checksum value for data in that data segment fragment; and

17 (3) determining a checksum value for all of the data in that
18 packet as a function of the retrieved and calculated checksum values.

19
20 6. The method of claim 5 further comprising the steps of:

21 for at least one packet in step (d), caching one or more of said calculated
22 checksum values for the data segment fragments in that packet, wherein the cached
23 checksum values can be used for determining a checksum value for a subsequent data
24 packet.

25
26 7. The method of claim 6 wherein:

27 for at least one data segment fragment in each of one or more of said set
28 of packets in step (d), the step of calculating a checksum value for that data segment

1 fragment in step (d)(2) further includes the steps of calculating the checksum value for
2 that data segment fragment as a function of one or more previously cached checksum
3 values.

4
5 8. The method of claim 3, wherein:
6 each packet in a set of said packets includes: (i) at least one complete
7 data segment, and (ii) a fragment of each of one or more data segments; and
8 the step (d) for providing a checksum value for data in each of the packets
9 in said set of packets further includes the steps of:

10 (1) retrieving the checksum value for said at least one complete
11 data segment in that packet from a corresponding entry in the checksum list;

12 (2) for each of said one or more data segment fragments in the
13 packet, retrieving the checksum value from the entry in the checksum list corresponding
14 to the data segment of which that data segment fragment is a part, and calculating a
15 checksum value for data in a complementary fragment of that data segment; and

16 (3) determining a checksum value for all of the data in that
17 packet as a function of the retrieved and calculated checksum values.

18
19 9. The method of claim 8, wherein step (d)(3) includes the steps of:
20 calculating an intermediate checksum value as a function of the retrieved
21 checksum values; and

22 adjusting the intermediate check sum value using the calculated
23 checksum values to provide the checksum value for all of the data in that packet.

24
25 10. The method of claim 8 further comprising the steps of:
26 for at least one packet in step (d), caching one or more of said calculated
27 checksum values in step (d)(2), wherein the cached checksum values can be used for
28 determining a checksum value for a subsequent data packet.

1 11. The method of claim 10 wherein:

2 for at least one data segment fragment in each of one or more of said set
3 of packets in step (d), step (d)(2) further includes the steps of calculating the checksum
4 value for said complementary fragment as a function of one or more previously cached
5 checksum values.

6
7 12. The method of claim 3, wherein:

8 each packet in a set of said packets includes: (i) at least one complete
9 data segments, and (ii) a fragment of each of one or more data segments; and
10 the step (d) for providing a checksum value for each of the packets in said
11 set of packets further includes the steps of:

12 (1) retrieving the checksum value for said at least one complete
13 data segment in that packet from a corresponding entry in the checksum list;

14 (2) for each of said one or more data segment fragments in the
15 packet, if that data segment fragment is smaller in size than a complementary fragment
16 of the data segment of which that data segment fragment is a part, then calculating a
17 checksum value for data in that data segment fragment, otherwise, retrieving the
18 checksum value from the entry in the checksum list corresponding to said data segment
19 of which that data segment fragment is a part, and calculating a checksum value for
20 data in said complementary fragment of that data segment; and

21 (3) providing a checksum value for the data in the packet as a
22 function of said retrieved checksum values and said calculated checksum values.

23
24 13. The method of claim 12 further comprising the steps of:

25 for at least one packet in step (d), caching one or more of said calculated
26 checksum values in step (d)(2), wherein the cached checksum values can be used for
27 determining a checksum value for a subsequent data packet.

1 14. The method of claim 13, wherein for at least one data segment fragment
2 in each of one or more of said set of packets in step (d), step (d)(2) further includes the
3 steps of:

4 if that data segment fragment is smaller in size than a complementary
5 fragment of the data segment of which that data segment fragment is a part, then
6 calculating a checksum value for data in that data segment fragment as a function of
7 one or more previously cached checksum values;

8 otherwise, retrieving the checksum value from the entry in the checksum
9 list corresponding to said data segment of which that data segment fragment is a part,
10 and calculating a checksum value for data in said complementary fragment of that data
11 segment as a function of one or more previously cached checksum values.

12
13 15. The method of claim 3, wherein:

14 each packet in a set of said packets includes a fragment of each of one or
15 more data segments;

16 the steps of providing a checksum value for each of the packets in said
17 set of packets further includes the steps of:

18 for each fragment of a data segment in that packet, calculating a
19 checksum value for data in that data segment fragment; and

20 determining a checksum value for all of the data in that packet as a
21 function of the calculated checksum values.

22
23 16. The method of claim 15, wherein for at least one data segment fragment,
24 the step of calculating a checksum value includes the steps of:

25 retrieving the checksum value from the entry in the checksum list
26 corresponding to the data segment of which that data segment fragment is a part;

27 calculating a checksum value for data in a complementary fragment of
28 that data segment; and

1 determining the checksum value for that data segment fragment as a
2 function of: (i) said retrieved checksum value, and (ii) said calculated checksum value
3 for data in the complementary fragment.
4

5 17. A storage device comprising:

6 (a) storage media for storing data segments;
7 (b) buffer memory; and
8 (c) a checksum circuit for providing checksum values for data
9 segments retrieved from the storage media for transfer into the buffer memory, and for
10 storing the checksum values in a checksum list including a plurality of entries
11 corresponding to the data segments stored in the buffer memory, each entry for storing
12 a checksum value for a corresponding data segment stored in the buffer memory;

13 wherein, for transferring packets of data out of the buffer memory, a checksum
14 value can be calculated for data in each packet based on one or more checksum
15 values stored in the checksum list.
16

17 18. The storage device of claim 17, wherein the checksum circuit further
18 comprises:

19 (i) a logic circuit for calculating a checksum value for a data segment;
20 (ii) means for locating an entry in the checksum table corresponding to
21 that data segment; and

22 (iii) means for storing the checksum value in the located entry in the
23 checksum list.
24

25 19. The storage device of claim 17, further comprising a microcontroller
26 configured by program instructions for building packets of data for transfer out of the
27 buffer memory, and for providing a checksum value for data in each packet based on
28 one or more checksum values stored in the checksum list.

1 20. The storage device of claim 19, wherein the microcontroller is further
2 configured by program instructions for building packets of data for transfer out of the
3 buffer memory, each packet in a set of said packets including one or more complete
4 data segments, and for providing a checksum value for each of the packets in said set
5 of packets by retrieving the checksum value for each data segment in that packet from
6 a corresponding entry in the checksum list and calculating a checksum value for that
7 packet as a function of the retrieved checksum values.
8

9 21. The storage device of claim 19, wherein the microcontroller is further
10 configured by program instructions for:

11 building packets of data for transfer out of the buffer memory, each packet
12 in a set of said packets including one or more complete data segments, and a fragment
13 of each of one or more data segments; and

14 providing a checksum value for each of the packets in said set of packets
15 by:

16 (i) retrieving the checksum value for each of said one or more
17 complete data segments in that packet from a corresponding entry in the checksum list;

18 (ii) for each data segment fragment in that packet, calculating a
19 checksum value for data in that data segment fragment; and

20 (iii) determining a checksum value for all of the data in that
21 packet as a function of the retrieved and calculated checksum values.
22

23 22. The storage device of claim 19, wherein the microcontroller is further
24 configured by program instructions for:

25 building packets of data for transfer out of the buffer memory, each packet
26 in a set of said packets including at least one complete data segment, and a fragment
27 of each of one or more data segments; and

28 providing a checksum value for data in each of the packets in said set of

1 packets by:

- 2 (i) retrieving the checksum value for said at least one complete
3 data segment in that packet from a corresponding entry in the checksum list;
4 (ii) for each of said one or more data segment fragments in the
5 packet, retrieving the checksum value from the entry in the checksum list corresponding
6 to the data segment of which that data segment fragment is a part, and calculating a
7 checksum value for data in a complementary fragment of that data segment; and
8 (iii) determining a checksum value for all of the data in that
9 packet as a function of the retrieved and calculated checksum values.

10
11 23. The storage device of claim 19, wherein the microcontroller is further
12 configured by program instructions for:

13 building packets of data for transfer out of the buffer memory, each packet
14 in a set of said packets including a fragment of each of one or more data segments;
15 and

16 providing a checksum value for each of the packets in said set of packets
17 further includes the steps of:

18 for each fragment of a data segment in that packet, calculating a
19 checksum value for data in that data segment fragment; and

20 determining a checksum value for all of the data in that packet as a
21 function of the calculated checksum values.

22
23 24. The storage device of claim 23, wherein the microcontroller is further
24 configured by program instructions such that for at least one data segment fragment,
25 the step of calculating a checksum value includes the steps of:

26 retrieving the checksum value from the entry in the checksum list
27 corresponding to the data segment of which that data segment fragment is a part;
28 calculating a checksum value for data in a complementary fragment of

1 that data segment; and

2 determining the checksum value for that data segment fragment as a
3 function of: (i) said retrieved checksum value, and (ii) said calculated checksum value
4 for data in the complementary fragment.
5

6 25. A checksum system for a data processing system including at least one
7 data storage device and buffer memory, the checksum system being for providing
8 checksum values for data segments retrieved from the data storage device for transfer
9 into the buffer memory, the checksum system comprising:

10 (a) a checksum list including a plurality of entries corresponding to the
11 data segments stored in the buffer memory, each entry for storing a checksum value for
12 a corresponding data segment stored in the buffer memory;

13 (b) a logic circuit for calculating a checksum value for each data
14 segment;

15 (c) means for locating an entry in the checksum table corresponding to
16 that data segment; and

17 (d) means for storing the checksum value in the located entry in the
18 checksum list;

19 wherein, for transferring packets of data out of the buffer memory, a checksum
20 value can be calculated for data in each packet based on one or more checksum
21 values stored in the checksum list.
22

23 26. The checksum system claim 25 further comprising a processor configured
24 by program instructions for building packets of data for transfer out of the buffer
25 memory, and for providing a checksum value for data in each packet based on one or
26 more checksum values stored in the checksum list.
27
28

1 27. The checksum system of claim 26, wherein the processor is further
2 configured by program instructions for building packets of data for transfer out of the
3 buffer memory, each packet in a set of said packets including one or more complete
4 data segments, and for providing a checksum value for each of the packets in said set
5 of packets by retrieving the checksum value for each data segment in that packet from
6 a corresponding entry in the checksum list and calculating a checksum value for that
7 packet as a function of the retrieved checksum values.

8
9 28. The checksum system of claim 26, wherein the processor is further
10 configured by program instructions for:

11 building packets of data for transfer out of the buffer memory, each packet
12 in a set of said packets including one or more complete data segments, and a fragment
13 of each of one or more data segments; and

14 providing a checksum value for each of the packets in said set of packets
15 by:

16 (i) retrieving the checksum value for each of said one or more
17 complete data segments in that packet from a corresponding entry in the checksum list;

18 (ii) for each data segment fragment in that packet, calculating a
19 checksum value for data in that data segment fragment; and

20 (iii) determining a checksum value for all of the data in that
21 packet as a function of the retrieved and calculated checksum values.

22
23 29. The checksum system of claim 26, wherein the processor is further
24 configured by program instructions for:

25 building packets of data for transfer out of the buffer memory, each packet
26 in a set of said packets including at least one complete data segment, and a fragment
27 of each of one or more data segments; and

28 providing a checksum value for data in each of the packets in said set of

1 packets by:

2 (i) retrieving the checksum value for said at least one complete
3 data segment in that packet from a corresponding entry in the checksum list;

4 (ii) for each of said one or more data segment fragments in the
5 packet, retrieving the checksum value from the entry in the checksum list corresponding
6 to the data segment of which that data segment fragment is a part, and calculating a
7 checksum value for data in a complementary fragment of that data segment; and

8 (iii) determining a checksum value for all of the data in that
9 packet as a function of the retrieved and calculated checksum values.

Abstract

A method and apparatus for generating checksum values for data segments retrieved from a data storage device for transfer into a buffer memory, is provided. A checksum list is maintained to contain checksum values, wherein the checksum list includes a plurality of entries corresponding to the data segments stored in the buffer memory, each entry for storing a checksum value for a corresponding data segment stored in the buffer memory. For each data segment retrieved from the storage device: a checksum value is calculated for that data segment using a checksum circuit; an entry in the checksum list corresponding to that data segment is selected; the checksum value is stored in the selected entry in the checksum list; and that data segment is stored in the buffer memory. Preferably, the checksum circuit calculates the checksum for each data segment as that data segment is transferred into the buffer memory.

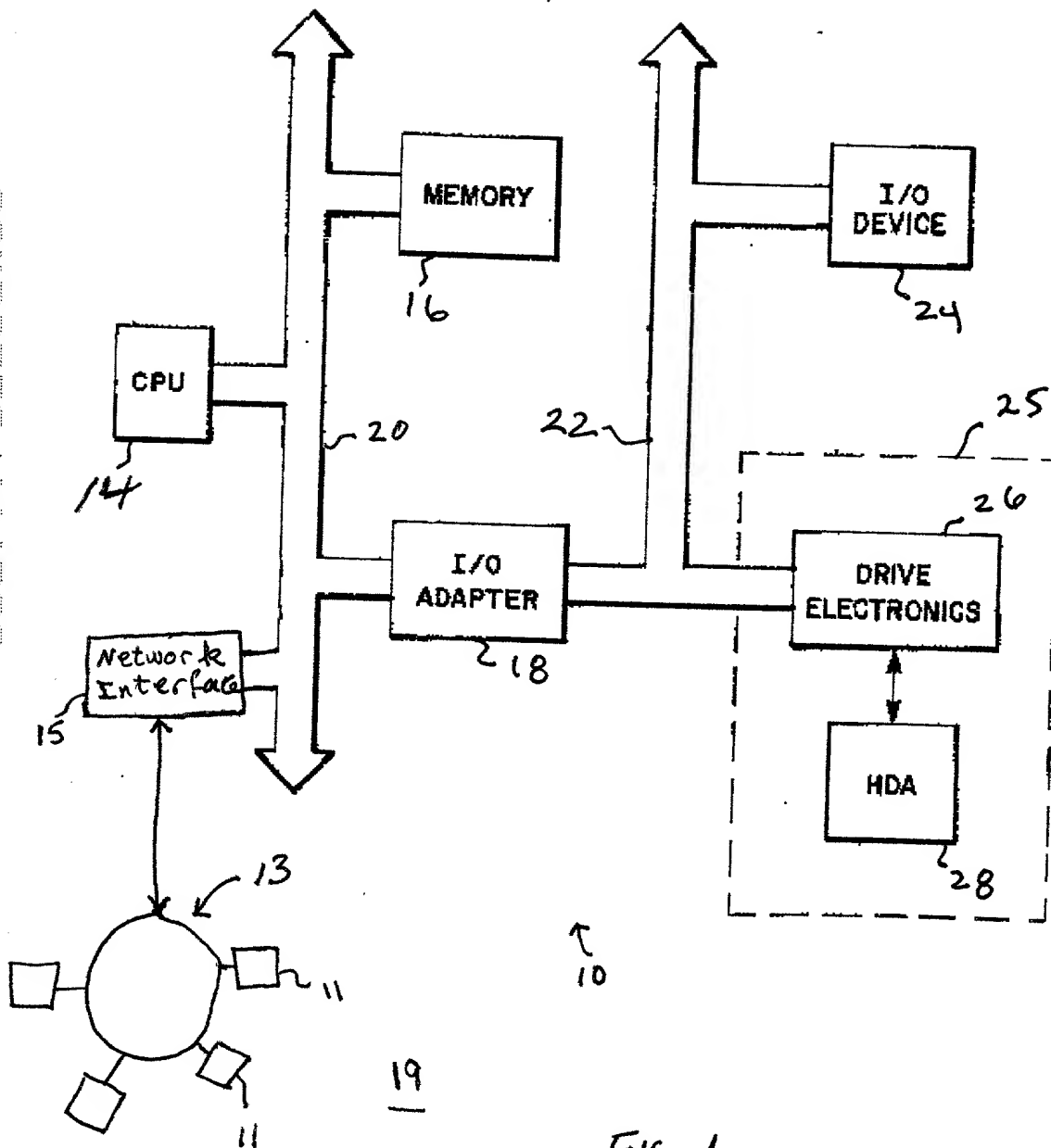


FIG. 1

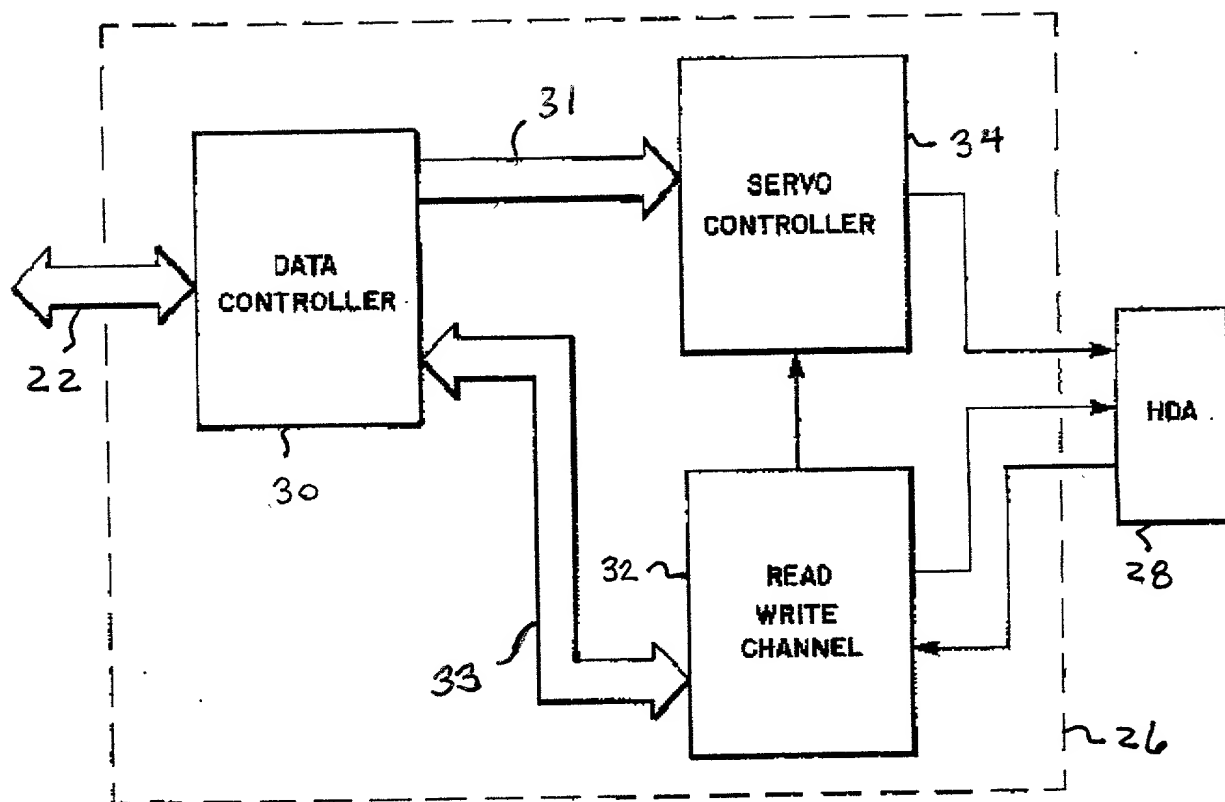
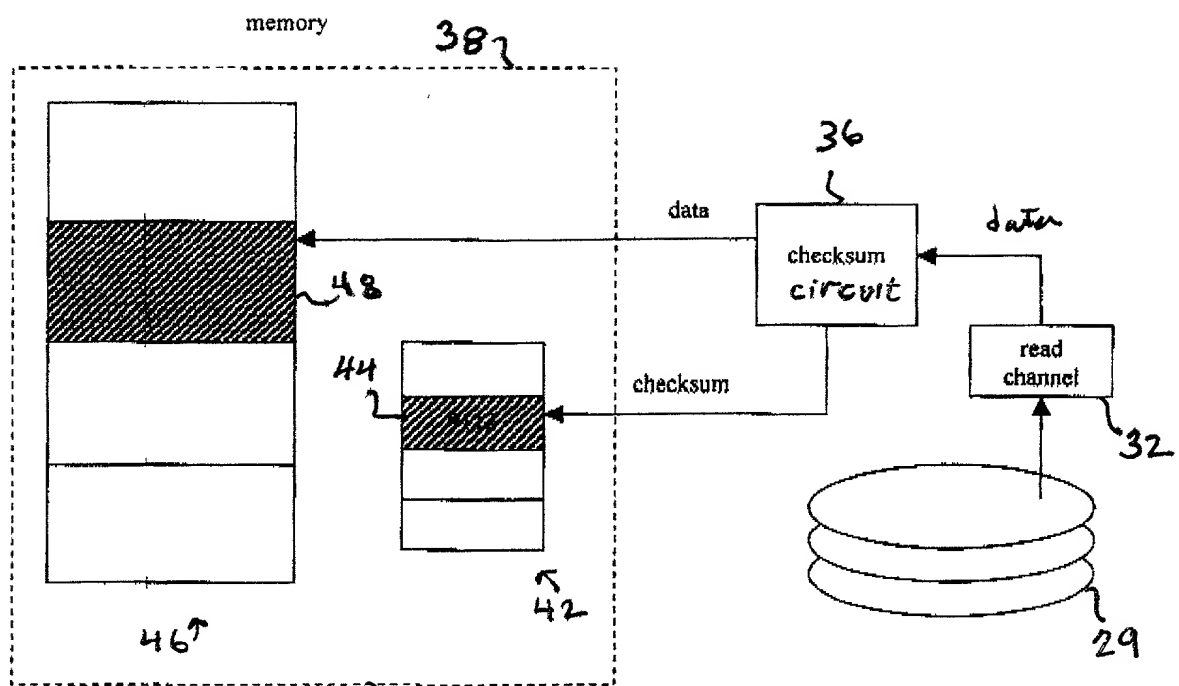


FIG. 2



30

FIG. 3

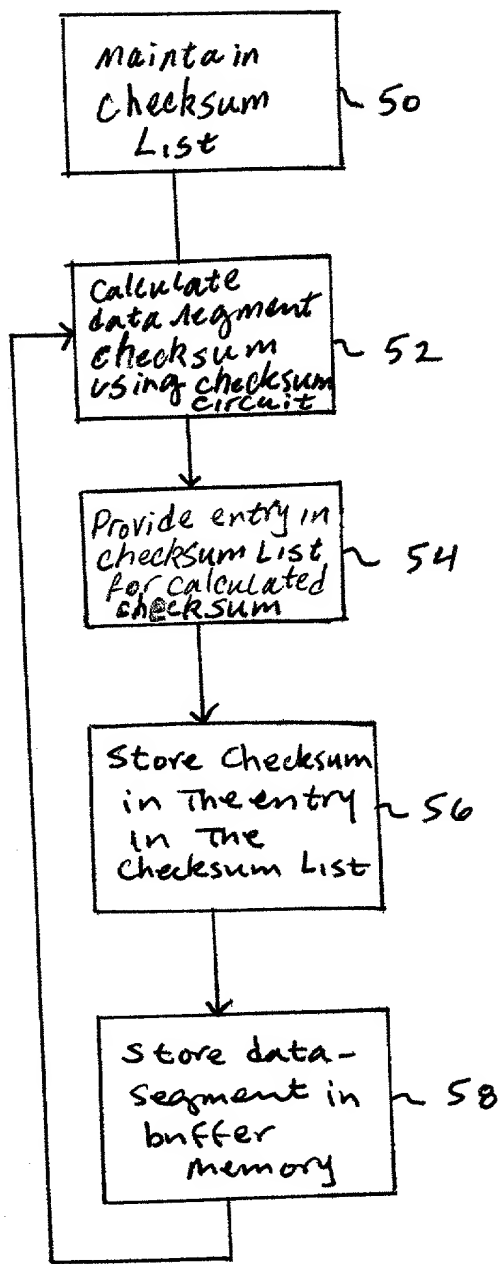


FIG. 4

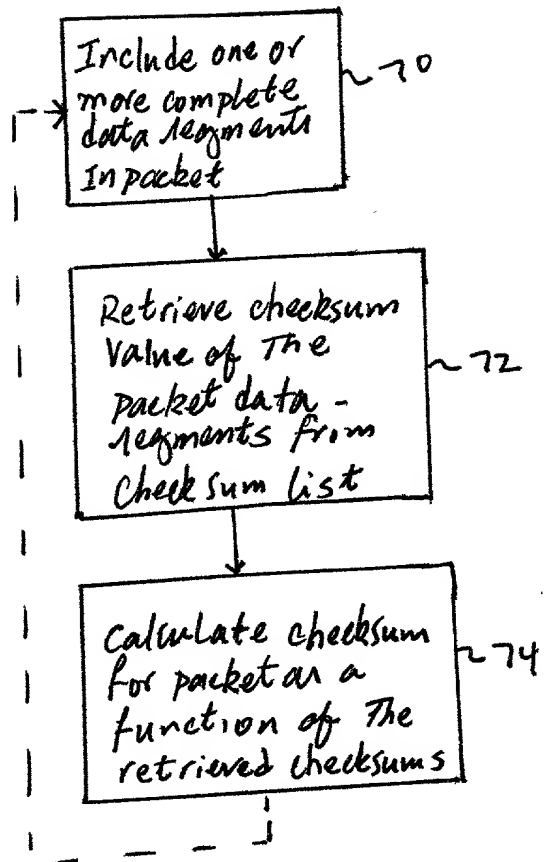


FIG. 6

662115926E460

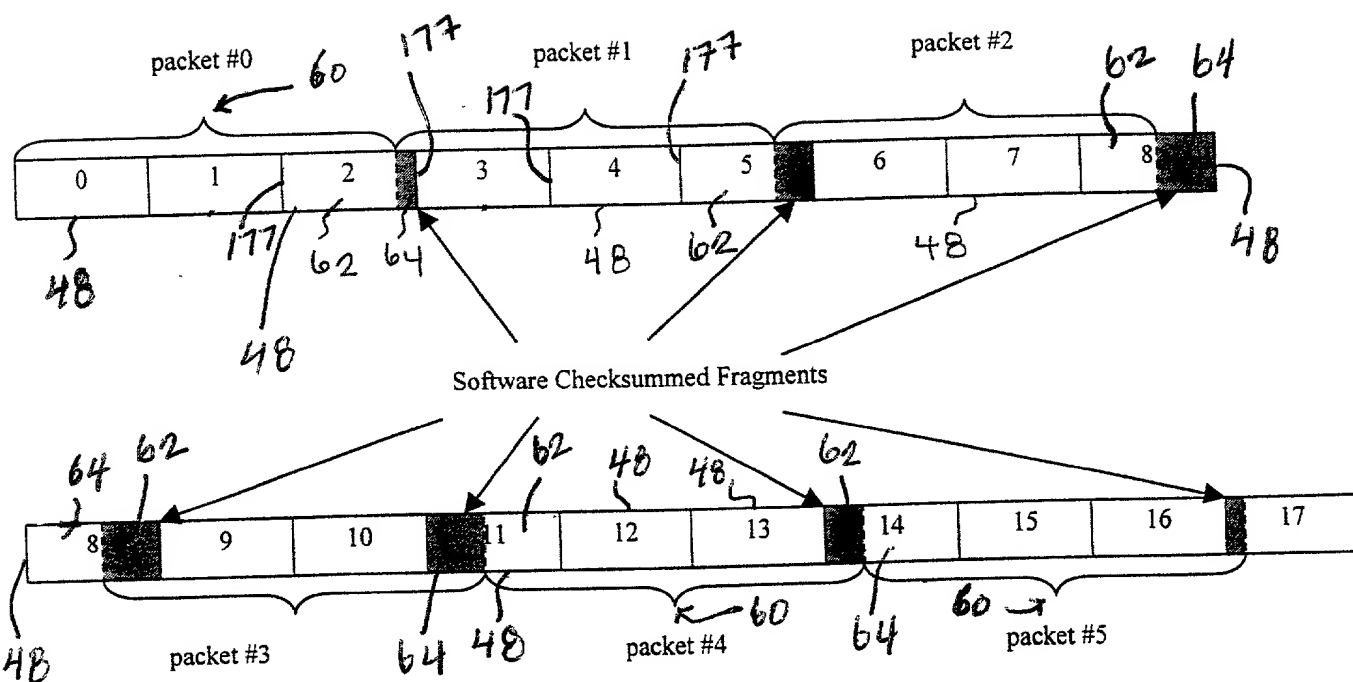


FIG. 5A

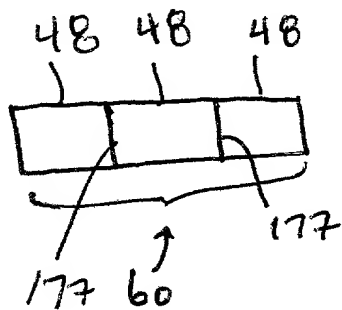


FIG. 5B

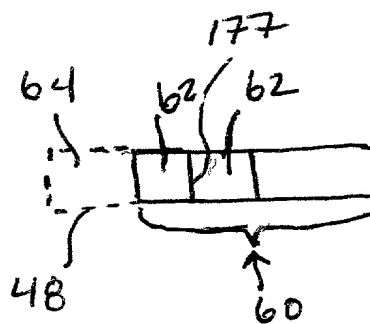


FIG. 5C

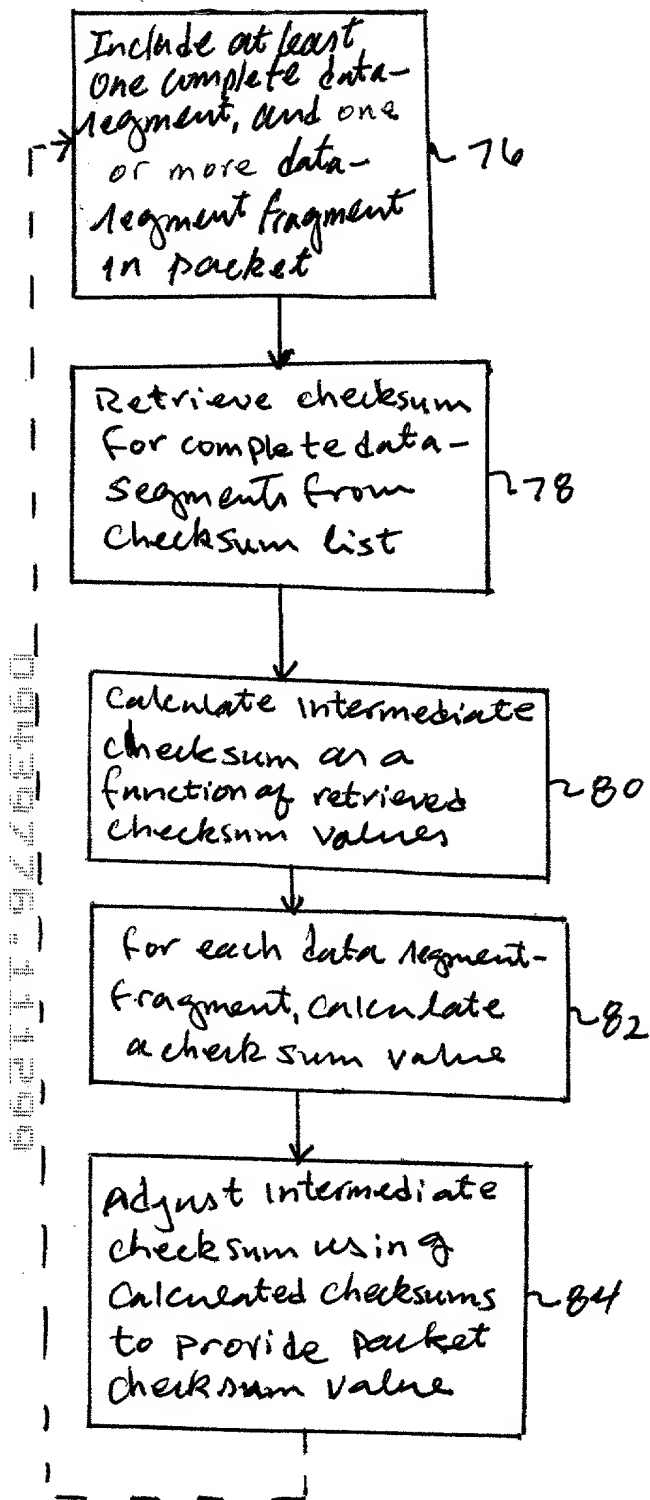


FIG. 7A

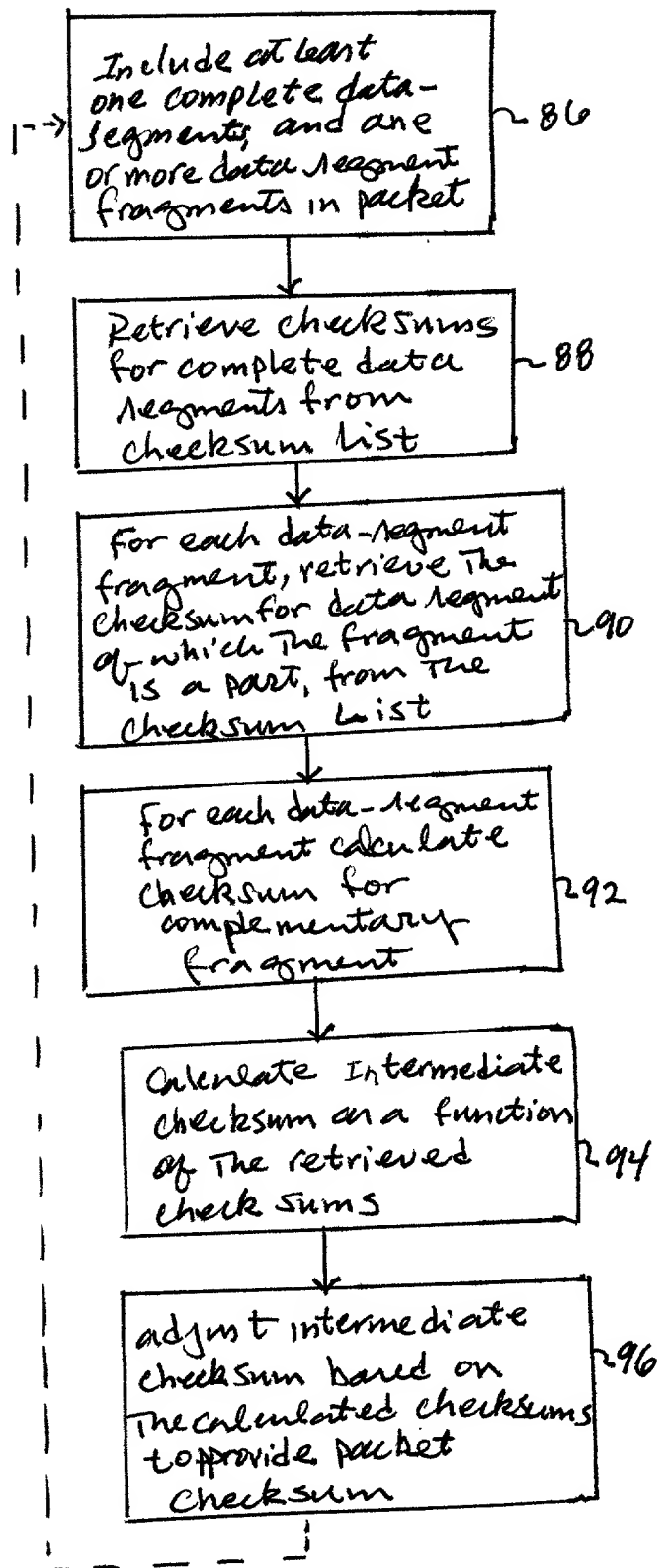


FIG. 7B

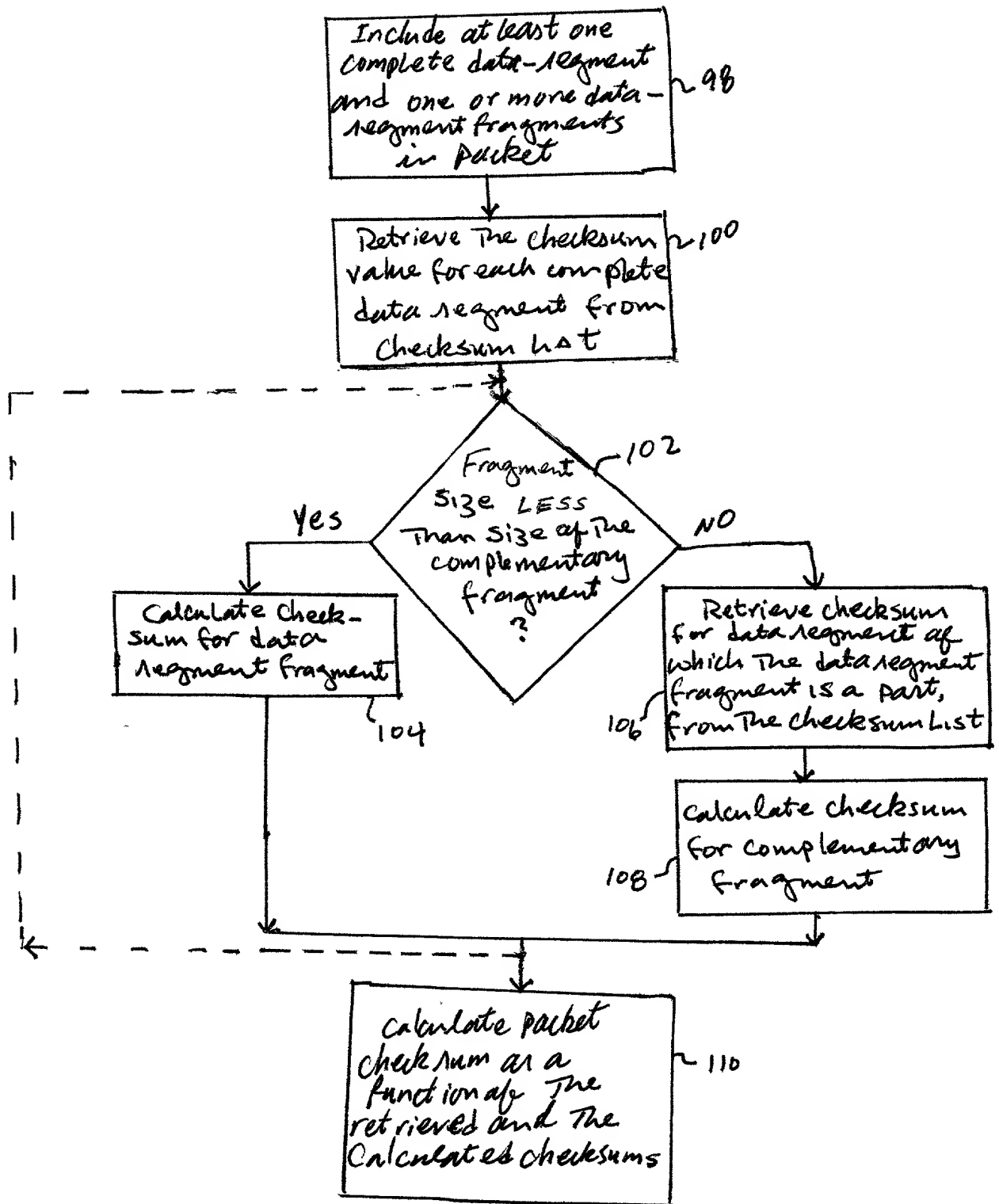


FIG. 7c

66211-926460

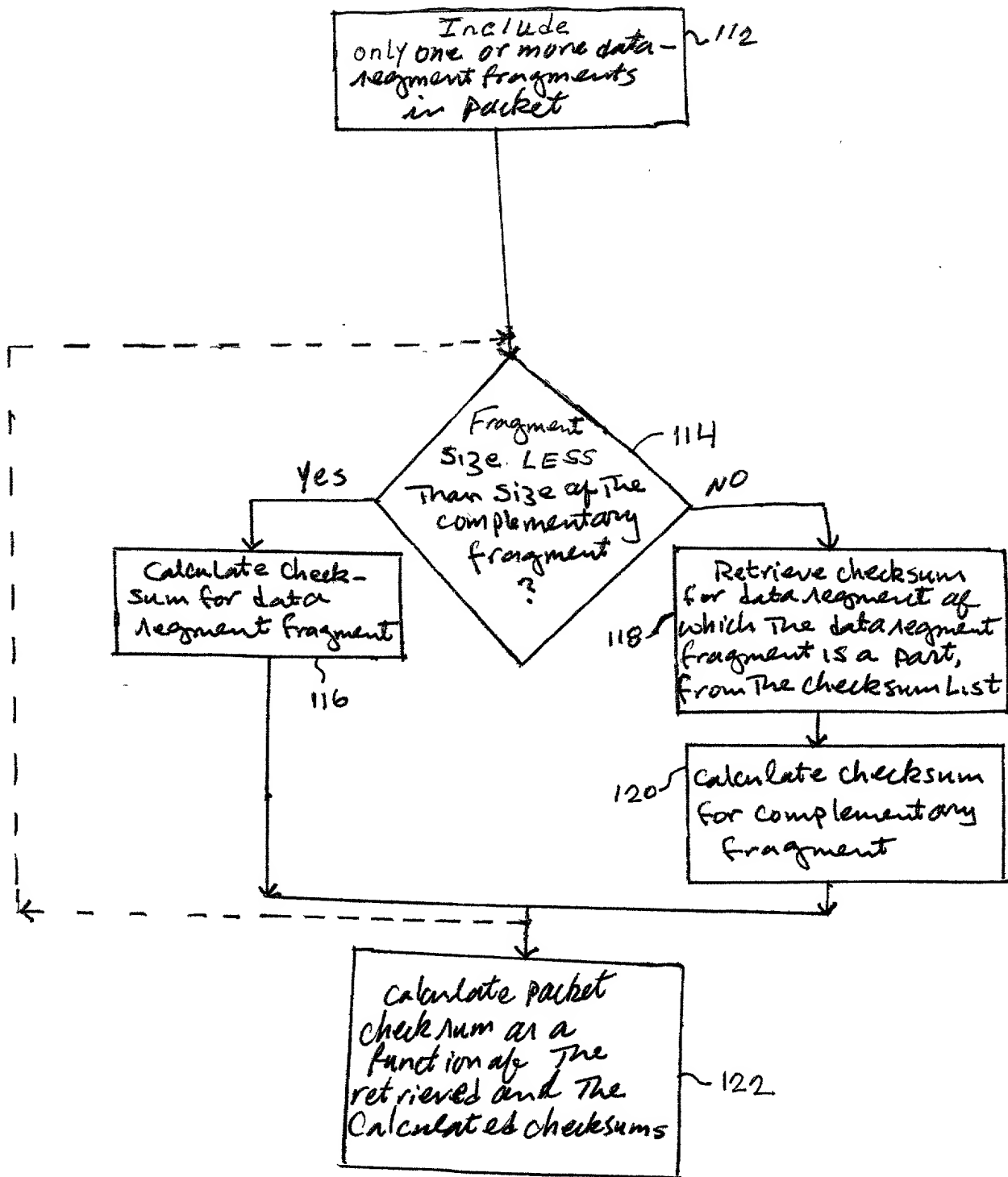


FIG. 8

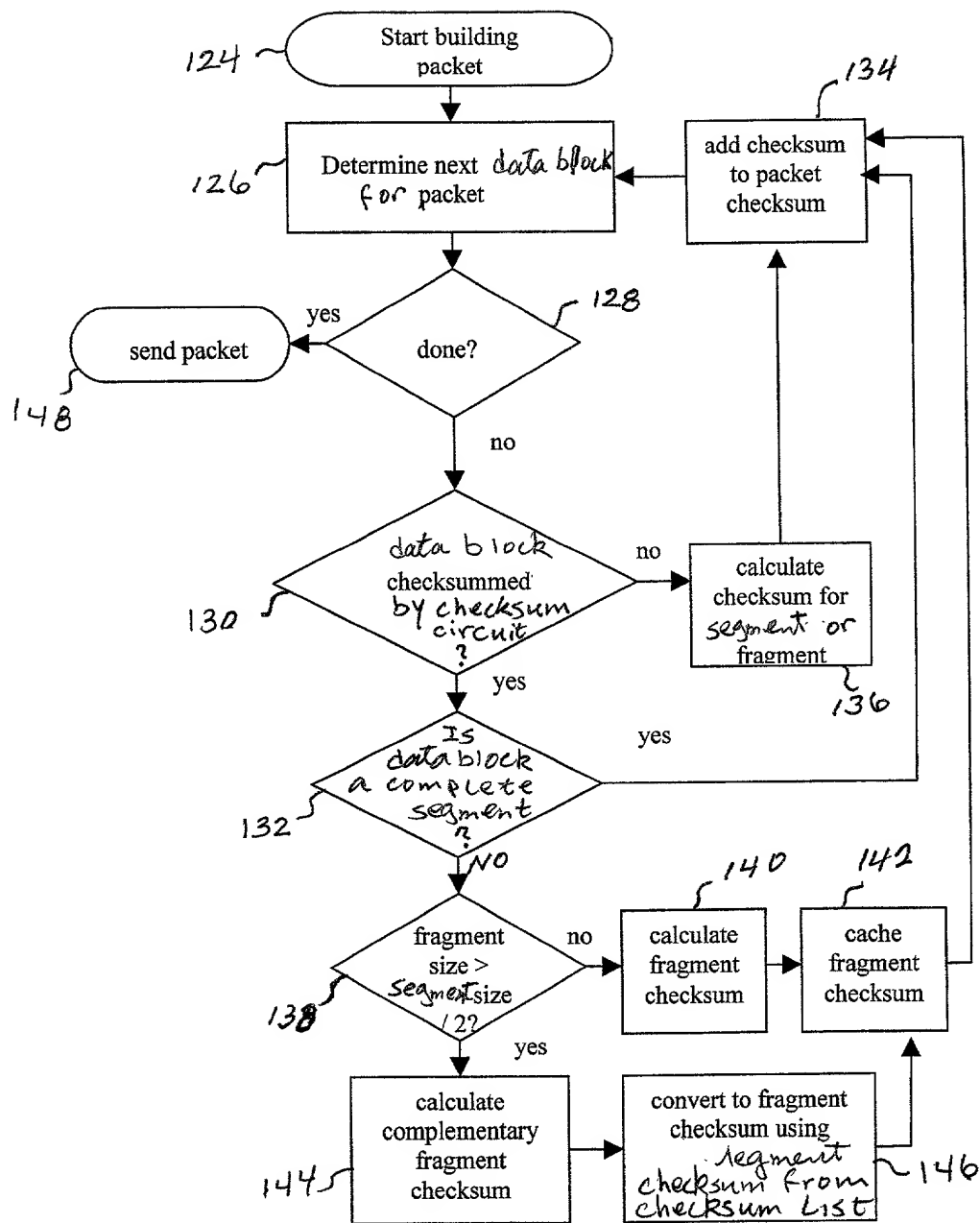


FIG. 9

662114-5226460

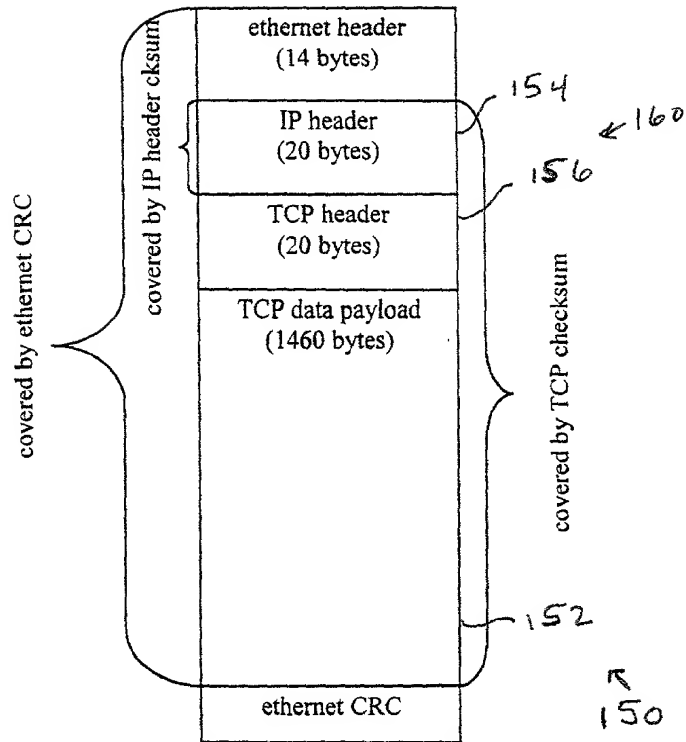


Fig. 10A

0	3	7	15	18	23	31
Version Number	Length	Service Type	Packet Length			
Identification				D M F F	Offset	
Time to Live		Transport		Header Checksum		
Source Address						
Destination Address						
Options (optional)						Padding

← 154

Fig. 10B

0	3	7	15	18	23	31
Source Address						
Destination Address						
Zero		Transport Protocol		Packet Length		

↑ 158

Fig. 10C

packet number	relative segment numbers	cumulative bytes	modulo 512	fragment software checksummed	software checksum size
0	0-2	1460	436	outer	76
1	2-5	2920	360	outer	152
2	5-8	4380	284	outer	228
3	8-11	5840	208	inner	208
4	11-14	7300	132	inner	132
5	14-17	8760	56	inner	56

FIG. 11

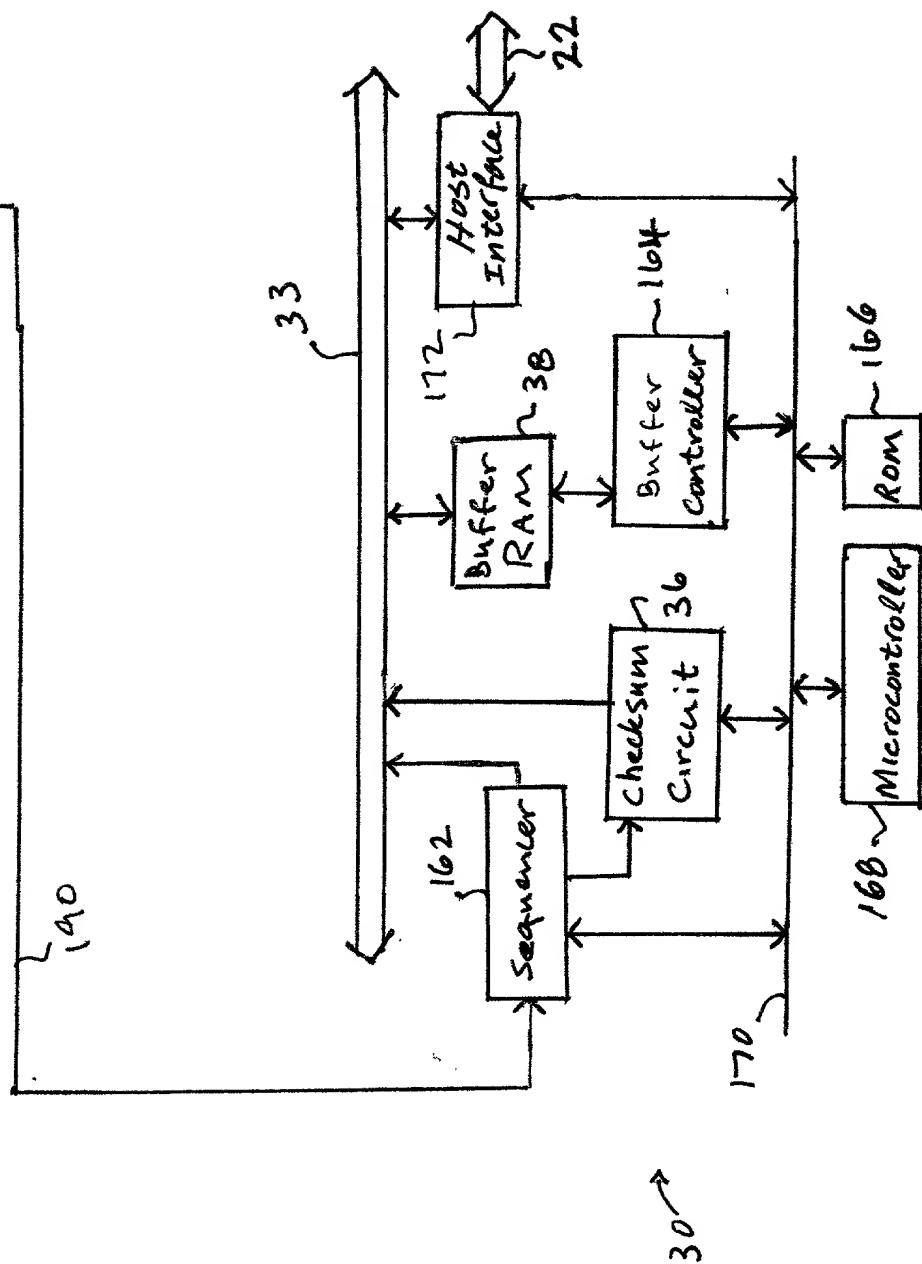
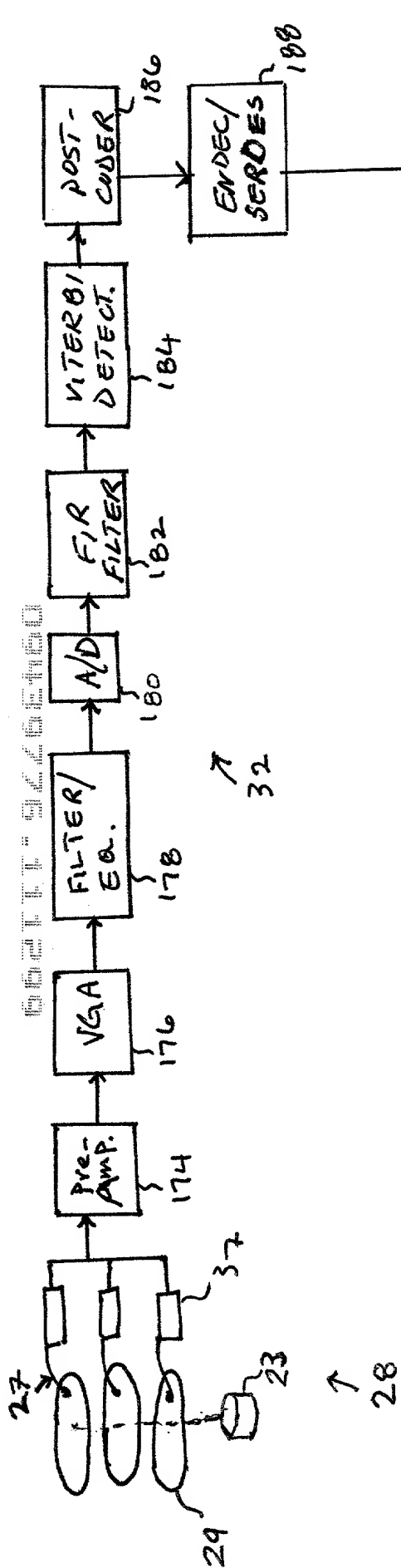


FIG. 12

662111-92266450

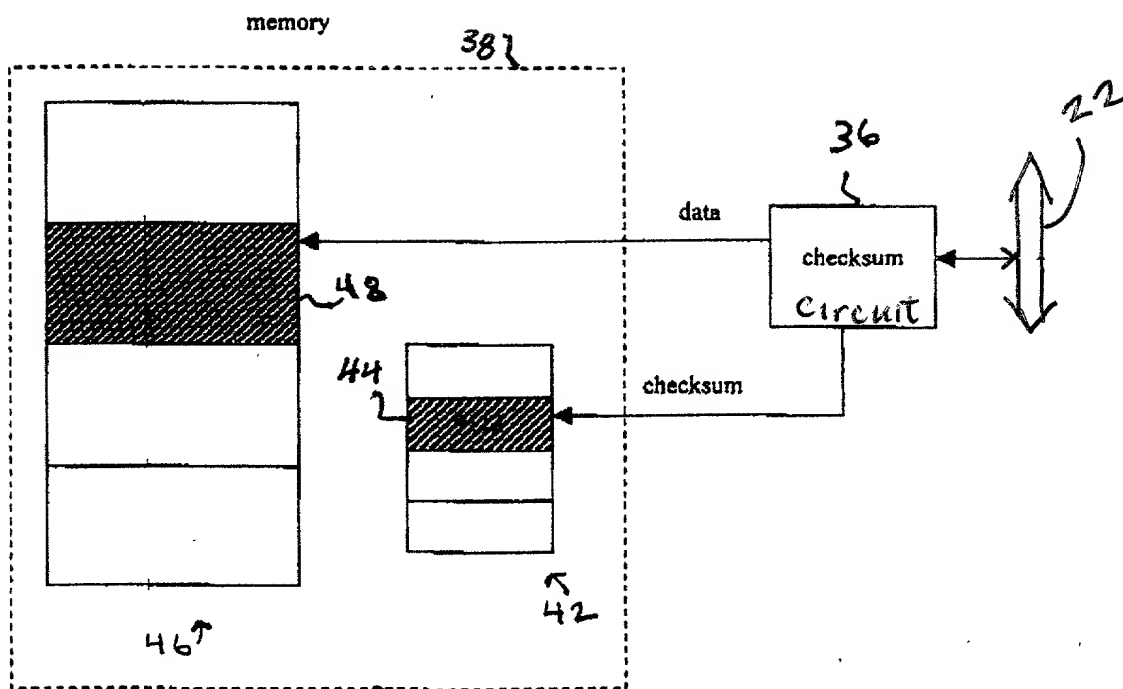


FIG. 14

66247-9/66460

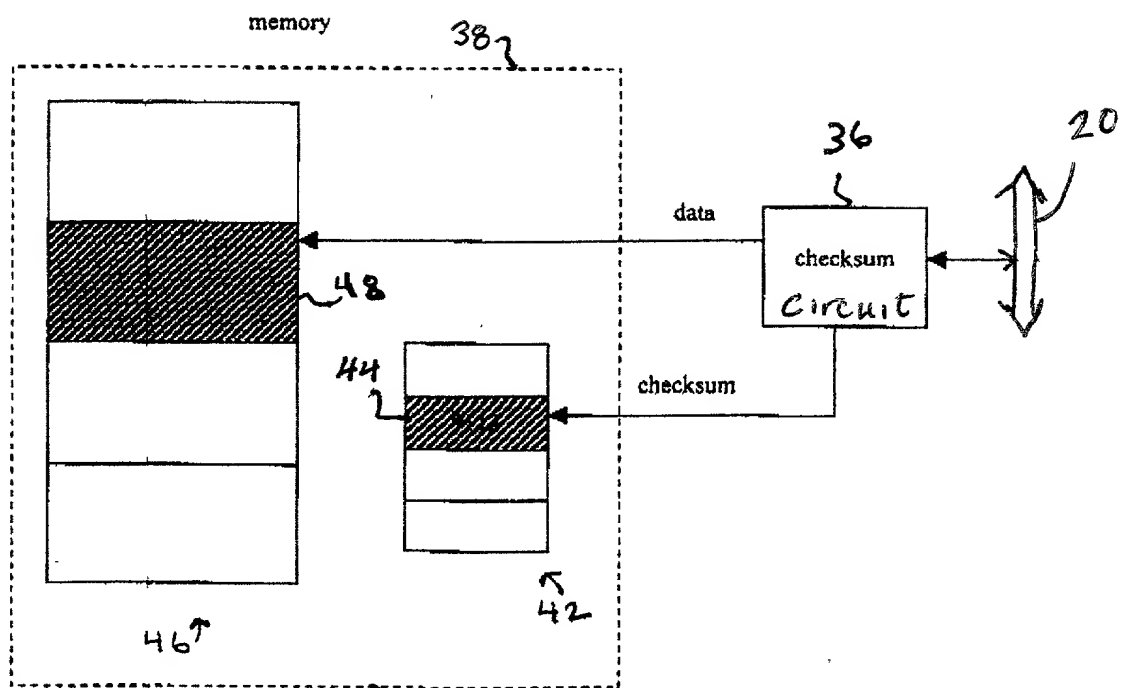


FIG. 15

DECLARATION FOR PATENT APPLICATION

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe that I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled "DATA CHECKSUM METHOD AND APPARATUS", the specification of which

 X is attached hereto.

 was filed on as Application Serial No.

and was amended by on .

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations, Section 1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code, Section 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Applications			Priority Claimed	
<u> </u>	<u> </u>	<u> </u>	<u> </u>	<u> </u>
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

_____	_____	_____	_____	_____
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Section 119(e) of any United States provisional application(s) listed below:

_____	_____
Provisional Application No.	Filing Date

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States applications(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application.

_____	_____	_____
(Application Serial No.)	(Filing Date)	(Status)
_____	_____	_____
(Application Serial No.)	(Filing Date)	(Status)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

On behalf of Quantum Corporation, Assignee of my entire right, title and interest, I hereby appoint the following attorney(s) and/or agent(s) with full power of substitution to act exclusively for Quantum Corporation to prosecute this application and transact all

business in the Patent and Trademark Office connected therewith: Jonathan B. Penn, Reg. No. 32,587; John C. Chen, Reg. No. 39,136; and Henry J. Groth, Reg. No. 39,696.

All correspondence should be addressed to:

John C. Chen
Patent Law Manager
Quantum Corporation
500 McCarthy Boulevard
Milpitas, California 95035

All telephone calls should be directed to __John C. Chen____, telephone number (408) _894-4191_.

Full Name of Sole Inventor: RODNEY VAN METER III

Inventor's Signature:  Date: 10/15/89

Inventor's Residence: 40 W. 4th Avenue, San Mateo, California 94402

Citizenship: United States of America

Post Office Address: Same as residence address above.

**POWER OF ATTORNEY BY ASSIGNEE
AND EXCLUSION OF INVENTOR(S) UNDER 37 C.F.R. ¶ 3.71 AND
CERTIFICATION UNDER 37 C.F.R 3.73(b)**

Hon. Commissioner of Patents and Trademarks
Washington, D.C. 20231

Dear Sir:

1. The undersigned assignee of the entire interest in the application for Letters Patent for the invention entitled: DATA CHECKSUM METHOD AND APPARTUS

 X attached hereto

 Serial No. Filed:

hereby appoints JONATHAN B. PENN, Registration No. 32,587, JOHN C. CHEN, Registration No. 39,136, and HENRY J. GROTH, Registration No. 39,696, its attorneys and/or agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith, said appointment to be to the exclusion of the inventor(s) and his or her attorney(s) in accordance with the provisions of 37 C.F.R. ¶ 3.71.

Please direct all communications relative to this application to the following addressee:

John C. Chen
Quantum Corporation
500 McCarthy Blvd.
Milpitas, California 95035

Telephone (408) 894-4191
FAX (408) 324-7005

2. Quantum Corporation, a Delaware corporation, certifies that it is the assignee of the entire right, title and interest in the patent application identified above by virtue of either:

 A. An assignment from the inventor(s) of the patent application identified above. The assignment was recorded in the Patent and Trademark Office at Reel , Frame , or for which a copy thereof is attached.

OR

X Copies of assignments or other documents in the chain of title are enclosed herewith.

The undersigned, whose title is supplied below, is empowered to sign this certificate on behalf of the assignee.

QUANTUM CORPORATION

By:

Andrew Kryder
Assistant Secretary

Appendix A

```
/*
 * INET      An implementation of the TCP/IP protocol suite for the LINUX
 *           operating system.  INET is implemented using the BSD Socket
 *           interface as the means of communication with the user level.
 *           IP/TCP/UDP checksumming routines.
 *           Code from tcp.c and ip.c.
 *           Free software; can be redistributed and/or
 *           modified under the terms of the GNU General Public License
 *           as published by the Free Software Foundation; either version
 *           2 of the License, or any later version.
 */
```

```
#include <asm/errno.h>
```

```
/*
 * computes a partial checksum, e.g. for TCP/UDP fragments
 */
```

```
/*
unsigned int csum_partial(const unsigned char * buff, int len, unsigned int sum)
*/
```

```
.text
.align 4
.globl csum_partial
```

```
/*
 * In Ethernet and SLIP connections, buff
 * is aligned on either a 2-byte or 4-byte boundary.  Provides at
 * least a twofold speedup on 486 and Pentium if it is 4-byte aligned.
 * 2-byte alignment can be converted to 4-byte
 * alignment for the unrolled loop.
 */
```

```
csum_partial:
    pushl %esi
    pushl %ebx
    movl 20(%esp),%eax # Function arg: unsigned int sum
    movl 16(%esp),%ecx # Function arg: int len
    movl 12(%esp),%esi # Function arg: unsigned char *buff
    testl $2, %esi     # Check alignment.
    jz 2f              # Jump if alignment is ok.
    subl $2, %ecx      # Alignment uses up two bytes.
    jae 1f             # Jump if we had at least two bytes.
    addl $2, %ecx      # ecx was < 2. Deal with it.
```

1: jmp 4f
movw (%esi), %bx
addl \$2, %esi
addw %bx, %ax
adcl \$0, %eax

2: movl %ecx, %edx
shrl \$5, %ecx
jz 2f
testl %esi, %esi

1: movl (%esi), %ebx
adcl %ebx, %eax
movl 4(%esi), %ebx
adcl %ebx, %eax
movl 8(%esi), %ebx
adcl %ebx, %eax
movl 12(%esi), %ebx
adcl %ebx, %eax
movl 16(%esi), %ebx
adcl %ebx, %eax
movl 20(%esi), %ebx
adcl %ebx, %eax
movl 24(%esi), %ebx
adcl %ebx, %eax
movl 28(%esi), %ebx
adcl %ebx, %eax
leal 32(%esi), %esi
dec %ecx
jne 1b
adcl \$0, %eax

2: movl %edx, %ecx
andl \$0x1c, %edx
je 4f
shrl \$2, %edx # This clears CF

3: adcl (%esi), %eax
leal 4(%esi), %esi
dec %edx
jne 3b
adcl \$0, %eax

4: andl \$3, %ecx
jz 7f
cmpl \$2, %ecx
jb 5f
movw (%esi), %cx
leal 2(%esi), %esi
je 6f
shll \$16, %ecx


```
5:    movb (%esi),%cl
6:    addl %ecx,%eax
    addl $0, %eax
7:
    popl %ebx
    popl %esi
    ret
```

Case 1:13-cv-00001-01

Appendix B

```
// Code to reblock data segments as packets and calculate checksums for packets
//
// copyright 1999, Quantum Corp.
// author Rodney Van Meter

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

// #include "/usr/src/linux/include/linux/linkage.h"
// #include "/usr/src/linux/include/asm-i386/checksum.h"

// below code is based on the above two files in the Linux source code

/*
 * computes the checksum of a memory block at buff, length len,
 * and adds in "sum" (32-bit)
 *
 * returns a 32-bit number suitable for feeding into itself
 * or csum_tcpudp_magic
 *
 * this function must be called with even lengths, except
 * for the last fragment, which may be odd
 *
 * preferably buff is aligned on a 32-bit boundary
 */

unsigned int csum_partial(const unsigned char * buff, int len, unsigned int sum);

/*
 * Fold a partial checksum
 */

static inline unsigned int csum_fold(unsigned int sum)
{
    __asm__(
        "addl %1, %0\n"
        "adcl $0xffff, %0\n"
        : "=r" (sum)
        : "r" (sum << 16), "0" (sum & 0xffff0000)
        );
    return (~sum) >> 16;
}
```

// end of code that is based on said two files in the Linux source code

```
#define BUFSIZE 512
#define NUMBUFS 200
#define PKTSIZE 1460
```

```
#define MIN(a,b) ((a) < (b)) ? (a) : (b)
```

```
char buffer[BUFSIZE*NUMBUFS];
```

```
// checksums contains intermediate 32-bit values that
// need to be folded together and complemented before sending in TCP
unsigned int checksums[NUMBUFS];
```

```
unsigned int Checksum(unsigned char *bufp, int len);
unsigned int ChecksumAdd(unsigned int a, unsigned int b);
unsigned int ChecksumSubtract(unsigned int a, unsigned int b);
```

```
main(int argc, char *argv[])
```

```
{
    int fd;
    int BufferEntry = 0;
    int TotalRead = 0;
    int TotalSent = 0;
    int CachedFragment = -1;
    unsigned int CachedFragmentCksum;    // always inner fragment cksum kept
    int CFSIZE;                         /* how much of the buffer? */
    int offset = 0;
    unsigned int ThisCksum = 0;
    unsigned int retval;
    int i;
```

```
    if ((fd = open(argv[1], O_RDONLY)) < 0) {
        perror("right up front");
        exit(-1);
    }
```

```
    // basic two-pass operation;
    // first pass is to read all of the data into memory;
    // assumes the read always returns exactly the amount requested
    while (TotalRead < BUFSIZE*NUMBUFS) {
        retval = ReadWithChecksum(fd,           // file descriptor
                                &buffer[BufferEntry*BUFSIZE], // buffer addr
                                BUFSIZE,        // size of read
                                &checksums[BufferEntry]);      // put cksum here
```

```
if (retval < 0) {
    perror("on read");
    exit(-1);
}
```

```
// add to our total
TotalRead += retval;
```

```
if (retval != BUFSIZE)
    break;
BufferEntry++;
} // end of while loop
```

```
/*
 * The following performs data reblocking and segment checksum management,
 * in addition to the arithmetic for checksum addition
 * and subtraction.
 */
```

```
// second pass; now data is transmitted onto network
// using the checksums from above
// go just to the last full packet
```

```
while (TotalSent + PKTSIZE < TotalRead) {
    // build and send a single packet
    int PktSize;
    int bufno;
    int modulo;
    unsigned int tmpcs;
    int thisfragsize;
```

```
PktSize = 0;
ThisCksum = 0;
```

```
// assumes use of the whole fragment; a packet
// is never smaller than BUFSIZE (e.g. for this code)
// and always send a full packet
while (PktSize < PKTSIZE) {
    bufno = (TotalSent + PktSize) / BUFSIZE;
    tmpcs = checksums[bufno];
    // modulo should be zero for every flag but the first
    modulo = (TotalSent + PktSize) % BUFSIZE;
    thisfragsize = MIN((PKTSIZE - PktSize), (BUFSIZE - modulo));
```

```
if (modulo == 0 &&
    thisfragsize == BUFSIZE) {
```

```
// whole (i.e. complete) data segment
ThisCksum = ChecksumAdd(ThisCksum, tmpcs);
printf("case a: whole segment checksum from hardware computed table\n");
} else {
    // only using a fragment, so calculate its cksum in software
    // and add it in, keeping the result cached in case the next
    // packet can use it
    if (CachedFragment == bufno) {
        // caching the single fragment won, now take advantage of it

        // this is equivalent to an error check
        if (CFSIZE != BUFSIZE - thisfragsize) {
            printf("Cannot use the cached fragment buf %d size %d for fragment size %d\n",
                bufno, CFSIZE, thisfragsize);
            goto mustdofrag;
        }

        // was inner trailing for prior packet, outer leading for this one
        tmpcs = ChecksumSubtract(checksums[bufno], CachedFragmentCksum);

        ThisCksum = ChecksumAdd(ThisCksum, tmpcs);
        printf("case b: using cached fragment checksum from prior packet\n");
    } else {
        mustdofrag:
        // cannot use cached checksum, so calculate checksum

        if (thisfragsize < BUFSIZE / 2) {

            // cksum this fragment
            tmpcs = Checksum(&buffer[TotalSent + PktSize], thisfragsize);
            ThisCksum = ChecksumAdd(ThisCksum, tmpcs);
            printf("case c: checksumming this fragment\n");
        } else {

            // cksum the complementary fragment
            unsigned int tmpcsfrag, actualcs;

            int offset;
            if (modulo) {
                // leading, so back up to beginning of this buffer
                offset = TotalSent - modulo;
            } else {
                // trailing, so go out to the end of the pkt
                // and run to the end of the buffer
                offset = TotalSent + PktSize + thisfragsize;
            }
        }
    }
}
```

```

    }

    tmpcsfrag = Checksum(&buffer[offset],
                        BUFSIZE - thisfragsize);

    // this converts to the inner (needed) fragment cksum
    tmpcs = ChecksumSubtract(checksums[bufno],tmpcsfrag);

    ThisCksum = ChecksumAdd(ThisCksum, tmpcs);
    printf("case d: checksumming the complementary fragment\n");
}

// now cache the inner trailing fragment checksum
CachedFragmentCksum = tmpcs;
CachedFragment = bufno;
CFSize = thisfragsize;
}
}

PktSize += thisfragsize;

} // end of while for checksum build

SendPacket(&buffer[TotalSent],PKTSIZE,ThisCksum);
TotalSent += PKTSIZE;
} // end of while full packet's worth of buffer left

// can additionally send final partial packet here;

// dump the table
for ( i = 0 ; i < NUMBUFS ; i++ ) {
    printf("%d: 0x%x\n",i,checksums[i]);
}
}

int ReadWithChecksum(int fd,          /* file descriptor */
                    char *bufp, /* buffer pointer to fill */
                    int size,      /* how many bytes should be read? */
                    int *cksump /* pointer to where to put the checksum */
                    )
{
    /* return value is the number of bytes read. Assumes read always returns
       what is requested for, unless it's the end of the file. */

    int retval;

    retval = read(fd,bufp,size);

```

```
*cksum = Checksum(bufp,size);
return retval;
}
```

```
unsigned int Checksum(unsigned char *bufp, int len)
{
    /* though not implemented here, a starting value can also be added */
    return csum_partial(bufp,len,0);
}
```

```
unsigned int ChecksumAdd(unsigned int a, unsigned int b)
{
    unsigned int retval;
    unsigned int carry;

    retval = a + b;
    carry = retval < a;
    retval += carry;

    return retval;
}
```

```
unsigned int ChecksumSubtract(unsigned int a, unsigned int b)
{
    // they are the same in ones-complement arithmetic
    unsigned int retval,tmp;

    retval = ChecksumAdd(a,~b);
    tmp = ChecksumAdd(retval,b);
    if (tmp != a) {
        printf("sub not sym? a: 0x%x b: 0x%x r: 0x%x r+b: 0x%x\n",
            a,b,retval,tmp);
    }
    return retval;
}
```

```
// this function prints out the 32-bit intermediate value
// calculated using the cached and logic circuit-generated checksums
// and double-checks it by recomputing the checksum on the whole packet
SendPacket(unsigned char *bufp, int len, unsigned int csum)
{
    unsigned int csum2;
    unsigned short folded,folded2;

    csum2 = Checksum(bufp,len);    /* double check */
    folded = csum_fold(csum);
    folded2 = csum_fold(csum2);
}
```

```
if (csum != csum2)
    printf("***** error!\n");
printf("snd %d: cksum: 0x%x -> 0x%x, doublecheck: 0x%x -> 0x%x\n",
    len,csum,folded,csum2,folded2);
}
```

00000000000000000000000000000000